Gustavo Okuyama 8504285

José Filipe Sanchez 8551538

SISTEMA DE GERENCIAMENTO DE EMPRESAS DISTRIBUIDORAS COM MÉTODO DE OTIMIZAÇÃO DE TRAJETO

23 de novembro de 2017

Gustavo Okuyama 8504285
José Filipe Sanchez 8551538

SISTEMA DE GERENCIAMENTO DE EMPRESAS DISTRIBUIDORAS COM MÉTODO DE OTIMIZAÇÃO DE TRAJETO

Trabalho de Conclusão de Curso

Prof. Dr. Evandro Luis Linhari Rodrigues
Departamento de Engenharia Elétrica e de Computação
Escola de Engenharia de São Carlos
Universidade de São Paulo

23 de novembro de 2017

Queremos dedicar o trabalho primeiramente a Deus, que nos criou e merece tudo de bom que façamos. Também as nossas famílias, amigos e nossos parceiros da Kitnet do seu Álvaro, que sempre nos apoiaram do começo ao fim desde que decidimos formar essa dupla. Dedicamos também a Valve que criou o magnífico Dota 2.

Agradecimentos

Agradecemos ao Professor Doutor Evandro Luis Linhari Rodrigues que executou muito bem seu trabalho como orientador. Ele nos deu muitas propostas e nos apresentou diversas ferramentas já existentes atualmente que puderam ser acrescidas ao trabalho para seu enriquecimento. Nos deu muitas ideias que podiam tornar nosso projeto ainda mais abrangente, mas nem todas puderam ser aplicadas por falta de tempo e experiência.

Queremos agradecer aos muitos companheiros da USP que, não só nos deram a ideia inicial para elaboração do projeto, mas também continuaram dando sugestões de melhoria e testaram nossos aplicativos desde suas primeiras versões.

Agradecemos também a todos os internautas que participam de fóruns de dúvidas na Web sobre programação. Muitas dos nossos problemas com programação foram sanados através desses fóruns e as soluções propostas foram em geral muito eficazes. Também àqueles programadores que disponibilizam seus códigos na Internet para uso. Muitas bibliotecas de código aberto foram utilizados nos nossos programas.

Eu, Gustavo, quero agradecer aos meus pais, João e Elenai, por terem me ajudado em toda minha trajetória, com amor, disciplina e carinho. Quero agradecer também à minha irmã Isabela e a minha namorada Sayumi que em todos esses anos me ajudaram e apoiaram. Agradeço também aos meus amigos da Engenharia da Computação pelo tempo que passamos juntos e por tudo o que fizeram por mim.

Eu, José Filipe, agradeço aos meus pais Mi e Elaine e minha querida noiva Vanessa por todo amor e ajuda que me deram nesse período. Durante esses 5 anos de estudos eles nunca deixaram de me apoiar e sempre ajudaram no que fosse preciso. Também aos amigos que fiz em São Carlos dentro e fora da faculdade. Apesar de toda a resistência que existe nos estudos, essa corrente de amigos me ajudaram a vencer toda a tensão de forma eletrizante.

"Errooooooou"
Fausto Silva

Resumo

Atualmente é muito comum as pessoas comprarem pela Internet. Isso já é uma prática tão comum e a tecnologia já evoluiu tanto nos dias atuais que elas desejam ter informações de suas encomendas em um clique. Também, querem ter uma base do momento certo em que a compra chega em sua casa. Mas, as empresas que prestam serviços de transporte não disponibilizam todas essas informações. Então, surge a ideia de criar um conjunto de aplicativos que resolvem o problema. O projeto envolve o desenvolvimento de aplicativos Android em conjunto com um servidor embarcado que auxilia no monitoramento de produtos comprados à distância e ainda gera a melhor rota de entrega a ser seguida pelos entregadores. Os aplicativos foram criados a partir do *software* Android Studio em linguagem Java. Foram utilizados diversas ferramentas para Java em auxílio, como métodos de manipulação de coordenadas GPS, suporte para comunicação com servidores, conversão de dados e leitor de códigos QR. O servidor foi desenvolvido pelo *framework* Django e executado em uma Raspberry Pi. Foi utilizado o Google Maps e suas ferramentas para trabalhar com mapas e também o Firebase Cloud Messaging para trabalhar com as notificações enviadas do servidor aos dispositivos móveis com informações da entrega. Os materiais necessários para a implantação do projeto não precisam ser sofisticados: os aplicativos são muito leves, gastam muito pouco da bateria dos *smartphones*, usam pouca memória RAM e foram projetados para Android, sistema presente na maioria dos dispositivos móveis e *tablets*; o servidor não precisa de tanta robustez: uma Raspberry Pi (bem simples) foi capaz de atender as necessidades sem problemas. O bom uso desse conjunto pode render muita economia por parte das empresas e uma grande satisfação dos seus clientes.

Palavras-chaves: *App Android*. Java. Monitoramento por GPS. Logística. Sistema Embarcado

Abstract

It is very common for people to buy online nowadays. This is already such a common practice and technology has evolved so much these days that they want to have information on their orders in one click. Also, they want to have a basis of the right moment when the purchase arrives in your house. But companies that provide transport services do not provide all this information. Then, the idea arises of creating a set of applications that solve the problem. This project involves the development of server embedded Android Apps for the purpose of monitoring orders purchased and generates the best route for the delivery driver. The Apps were developed using an Android Studio Software using Java programing language. Many Java tools were used in the development, such as GPS coordinate management, server communication support, data conversion, and a QR code reading scanner library. The server uses a Django web framework and runs on a Raspberry Pi single board computer. Google Maps was also used for work with maps. The project used Firebase Cloud Messaging service to work with push notifications sent by the server with delivery information. The materials needed to deploy the project need not be sophisticated: applications are very light, spend very little on the smartphone battery, require low computational cost and are designed for Android, a system that is present in most mobile devices; the server does not need much robustness: a RaspberryPi (very simple) was able to meet the needs without problems. The good use of this can lead to many economies on the part of the companies and the great satisfaction of their clients.

Keywords: Android App. Java. GPS. Monitoring. Best-route. Embedded System

Sumário

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
App	Aplication
BaaS	Backend-as-a-service
CRUD	Create Read Update Destroy
DBMS	DataBase Manegement System
DRY	Don't Repeat Yourself
E-commerce	Eletronic Commerce
EESC	Escola de Engenharia de São Carlos
FCM	Firebase Cloud Message
GB	Gigabyte
GPS	Global Positionament System
HDMI	High-Definition Multimedia Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IOT	Internet of Things
JSON	Java Script Object Notation
MB	Megabyte
MHz	Mega Hertz
microSD	Micro Secure Digital
MTV	Model Template View
QR	Quick Response
RAM	Random Access Memory
REST	Representational State Transfer

SBC	Single Board Computer
SDK	Software Development Kit
SoC	System on a Chip
URL	Uniform Resource Locator

1 Introdução

O mercado mundial está em constante evolução. Alguns anos atrás, por exemplo, ao fazer uma compra era necessário se locomover até a loja, escolher o produto e pagar por ele com dinheiro ou cheque. Porém, com o avanço da tecnologia, é possível comprar de qualquer lugar, desde que conectado a internet, utilizando um cartão de crédito ou até mesmo moeda virtual. No Brasil observa-se um número cada vez maior de pessoas comprando *online* e esta é uma tendência mundial.

Um setor que tem acompanhado o mercado eletrônico é o setor de transportes. Para uma compra realizada na Internet ser entregue até o cliente é necessário um complexo sistema de logística para gerenciar todas as encomendas. Para acompanhar o *e-commerce*, o setor de transportes deve evoluir para a satisfação do cliente e o consequente crescimento do mercado.

Além deste setores, outro setor que evoluiu foi o de eletrônicos, com destaque para os *smartphones*. Uma pesquisa realizada pela Fundação Getúlio Vargas (GVCIA, 2016) revelou que até o final de 2017 haverá a proporção de um por habitante, contabilizando apenas os aparelhos em uso. Os *smartphones* são celulares inteligentes que possuem funcionalidades como GPS, acelerômetro, giroscópio dentre outras. Versões mais recentes possuem recursos como medição de batimentos cardíacos, sensor de digital e até leitor de íris.

Outro ponto em destaque é o crescimento do setor de Internet. A pesquisa TIC de 2016 (NIC.BR, 2016) revelou que 68 % dos habitantes brasileiros utilizam a Internet, e que destes 93 % acessam através do celular.

1.1 Motivação

Uma pesquisa realizada em 2016 (PERES N.; OKUYAMA, 2016) a respeito do serviço de entrega das transportadoras revelou que a vaga informação fornecida sobre a entrega do produto incomoda muito os consumidores. Os métodos utilizados atualmente indicam apenas que o produto saiu da transportadora, e não informam em que horário o produto será entregue. Por exemplo, o cliente é informado que o produto saiu para entrega e que pode ser entregue das 8h às 18h, um intervalo de 10 horas. Parte dos consumidores não estão aptos a receber a encomenda em todo esse intervalo, porém conseguem receber em parte dele. Por esse motivo afirmam que o serviço das transportadoras deixam a desejar. Inclusive, alguns comentam que o produto sempre chega no único momento que não estão em casa.

Para solucionar este problema algumas empresas implementaram a entrega agendada. Neste tipo de entrega o cliente escolhe qual o horário e o dia em que o produto deve ser entregue. Porém as encomendas feitas sob agendamento demoram mais para chegar do que as outras, com diferenças de talvez semanas.

Verificado esse problema nas cidades brasileiras analisou-se uma forma de solucioná-lo usando as tecnologias já utilizadas nas empresas transportadoras, como o sistema de rastreamento. Através do GPS é possível obter a posição dos veículos através de coordenadas globais. Posteriormente, utilizá-las para avaliar seus funcionários e rastrear o caminhão em caso de furto.

Além disso, analisou-se quais ferramentas podem aprimorar o serviço de entrega. Por exemplo, existem ferramentas disponíveis que calculam o tempo para realizar determinado trajeto e a melhor rota entre diversos pontos. Utilizar destas rotas geram uma economia de tempo e recursos da empresa. Estes recursos podem ser utilizados para investir em tecnologia para a empresa. Além disso, realizar a rota mais rápida permite que mais entregas sejam realizadas em um dia, reduzindo o número de veículos em operação.

Procurou-se na internet quais grandes empresas de *e-commerce* utilizam serviços de entrega além do convencional ou do agendado em cidades do porte de São Carlos. Obteve-se como resultado que nenhuma empresa utiliza um serviço que avisa o cliente com precisão que hora o produto chegará em casa. Pesquisou-se por serviços relacionados e foi encontrado o serviço de *motoboy online* prestado pela empresa Loggi. Esse serviço, específico para a cidade de São Paulo, é utilizado por algumas das grandes *e-commerces*. A entrega das encomendas é realizada através de *motoboys*. O cliente recebe as informações de onde seu produto está e em que horário ele chegará. A empresa que contrata o serviço recebe status em tempo real e relatórios detalhados da entrega. Em avaliações na internet observou-se que, nesse modelo, os motoqueiros que prestam serviço para a Loggi não o fazem com qualidade. Isso gera limitações de escalabilidade, como garantir a qualidade de entrega em um número maior de cidades, problema existente em outros serviços parecidos, como Uber.

Portanto, é necessário um sistema que possa gerenciar as encomendas de uma transportadora avisando aos clientes em que horário o produto chegará. Um sistema que possa ser utilizado em qualquer cidade sem problemas de escalabilidade, aproveitando as tecnologias já utilizadas pela empresa.

1.2 Objetivos

Com os dados e ferramentas existentes deseja-se fazer um sistema que:

- Calcule o trajeto mais rápido para um conjunto de entregas;

- Utilize os dados de GPS, além de outras informações para calcular o horário de entrega;
- Avise ao consumidor, através do *smartphone*, a estimativa do horário de entrega da encomenda.

1.3 Justificativas

Os programas de gerenciamento utilizados pelas empresas de transporte estão atrasados e trazem muita insatisfação para os clientes. Já existem muitas ferramentas que podem trazer mais riqueza de informações, otimizações de custo e melhor gerenciamento das transportadoras.

1.4 Organização

O trabalho foi dividido em:

- Embasamento Teórico - são apresentadas pequenas definições teóricas em âmbito mais geral de elementos que aparecem no trabalho.
- Materiais - é apresentado tudo aquilo que compõe o projeto, tanto físico como digital.
- Métodos - é apresentado todo o mecanismo de funcionamento. Como os materiais trabalham entre si, como os programas criados funcionam e o modo correto de usá-los.
- Resultados e Discussões - este capítulo tem por finalidade validar o que foi feito. São apresentados testes e é feita uma comparação com o que era previamente esperado.
- Conclusão - Aqui há uma revisão geral do que foi feito, um julgamento dos resultados obtidos e sugestões para melhoras futuras.

2 Embasamento Teórico

Este capítulo tem como objetivo apresentar os conceitos necessários para entendimento do sistema desenvolvido.

2.1 Sistema Operacional

Conjunto de programas mais importante executados em um dispositivo. Faz a comunicação entre a parte física (*hardware*) e o *software*. Gerencia recursos do sistema como memória e armazenamento. Pode comunicar-se com outros dispositivos através de um protocolo (ex.: Internet). Exemplos de sistema operacional: Ubuntu, MacOS, Windows 10, Android.

2.2 Framework

Framework é uma abstração de código comum entre vários projetos provendo uma funcionalidade genérica. Um *framework* fornece uma ou mais soluções para uma família de problemas semelhantes utilizando um conjunto de classes e interfaces.

Um *framework* Web facilita e agiliza o desenvolvimento de páginas Web integrando recursos complexos com poucas linhas de código. Como exemplo de framework Web cita-se Django e Rails.

2.3 REST

Representational State Transfer (REST) tem como objetivo primário a definição de características fundamentais para a construção de aplicações. REST é frequentemente aplicado a *Web Services* fornecendo APIs para acesso a um serviço qualquer na Web. Ele usa integralmente as mensagens HTTP para se comunicar através do que já é definido no protocolo sem precisar inventar novos protocolos específicos para aquela aplicação.

Uma das propriedades do HTTP é o cabeçalho. Este transfere informações como o código de estado (lembre-se do famoso código 404). Em HTTP cada código representa um estado (para 404, página não encontrada). Quando um *Web service* tentar acessar uma página Web REST que não existe, esta retornará um HTTP com um cabeçalho indicando que o código de estado é 404. Se o *Web service* for programado seguindo os padrões, entenderá que o resultado daquela requisição foi página não encontrada. A tabela 1 indica um resumo das respostas.

Tabela 1 – Resumo das respostas de um servidor

Código	**Estado da Resposta**
1XX	Informações Gerais
2XX	Sucesso
3XX	Redirecionamento
4XX	Erro no cliente
5XX	Erro no servidor

Além disso, aplicações REST são focadas no recurso e não nas operações. Por exemplo: www.querovender.com/delete/produto/1 foca na operação "delete" e por isso não é REST. Um exemplo de REST é www.querovender.com/produto/1 utilizando o verbo DELETE. Neste caso foca-se o recurso (produto de id 1) e não a operação. Além disso, o segundo exemplo utiliza verbos (podem ser: GET, POST, PUT, DELETE), outra propriedade do protocolo HTTP (ALGAWORKS, 2016). A tabela 3 mostra um exemplo de aplicação REST para o recurso "fotos".

Tabela 2 – Interface HTTP para o recurso *photos* utilizada em REST

Verbo	**Caminho**	**Ação**	**Usado para**
GET	/fotos	*index*	Mostrar lista com todas as fotos
GET	/fotos/new	*new*	Retornar formulário HTML para criar nova foto
POST	/fotos	*create*	Cria nova foto
GET	/fotos/<id>	*show*	Mostra foto específica
GET	/fotos/<id>/edit	*edit*	Retornar formulário HTML para editar foto específica
PUT	/fotos/<id>	*update*	Atualiza foto específica
DELETE	/fotos/<id>	*destroy*	Deleta foto específica

Caso uma aplicação seja totalmente REST esta é conhecida com RESTful.

2.4 API

API é um conjunto de rotinas e padrões de programação para acesso a um aplicativo de software ou plataforma baseado na Web. A sigla API refere-se ao termo em inglês *Application Programming Interface* que significa em tradução para o português "Interface de Programação de Aplicativos".

Através das APIs, os aplicativos podem se comunicar uns com os outros sem conhecimento ou intervenção dos usuários. Elas funcionam através da comunicação de diversos códigos, definindo comportamentos específicos de determinado objeto em uma interface. A API liga as diversas funções em um site de maneira que possam ser utilizadas em outras aplicações.

2.5 Banco de Dados

Um banco de dados é uma coleção de dados inter-relacionados, representando informações sobre um domínio específico. Sempre que for possível agrupar informações que se relacionam e tratam de um mesmo assunto, pode-se definir como um banco de dados.

Caso um sistema possua um banco de dados é necessário um gerenciador para acessar, modificar e controlar os dados. Este gerenciador é conhecido como DBMS (*Database Management Systems*) em português Sistema de Gerenciamento de Banco de Dados (HELLMAN, 2016). Exemplo de sistemas DBMS são: MySQL, MariaDB, PostGreSQL.

2.6 App

App ou Aplicativo é um Software desenvolvido para um Sistema Operacional. É mais comum ouvir a expressão App ser usada para dispositivos móveis como *tablets* e celulares. Eles são programados para diversas funções, seja para processamento de dados, consultas à bancos, redes sociais, jogos, reprodução de mídias, etc.

2.7 Código QR

É um código de barras bidimensional, sendo uma forma de compactação de informação comumente utilizada para leitura em *smartphones*. Alguns dos dados não guardam informação em si, mas estão presentes apenas para correção de erros, possibilitando que seja lido mesmo com câmeras simples, como a de celulares.

Figura 1 – Exemplo de código QR

Pode armazenar números, frases, URLs, aumentando-se a compactação com o aumento de sua resolução (maior número de quadradinhos em seu interior). A capacidade máxima de armazenamento de um código QR ocorre na sua versão 40 nível L (imagem de 177 x 177 em que apenas 7% dos dados são usados para correção de erros). Ele pode armazenar 4296 caracteres alfanuméricos (QRCODE, 2017).

2.8 Notificações PUSH

São as notificações enviadas para usuários de determinado App de celular ou de outro aparelho. Eles têm como característica importante poder ser enviados mesmo com o App em *background* (não operando). Ao clicar na notificação, geralmente o usuário é direcionado ao aplicativo.

2.9 JSON

Java Script Object Notation é uma forma simples de se exprimir um objeto ou lista de objetos. Objeto é uma estrutura de dados que pode conter variáveis de diversas classes, como inteiros, reais, *strings*, booleanas ou até outros subobjetos. Todas as informações são armazenadas em uma grande *string* seguindo um certo padrão estabelecido, conforme a figura 2.

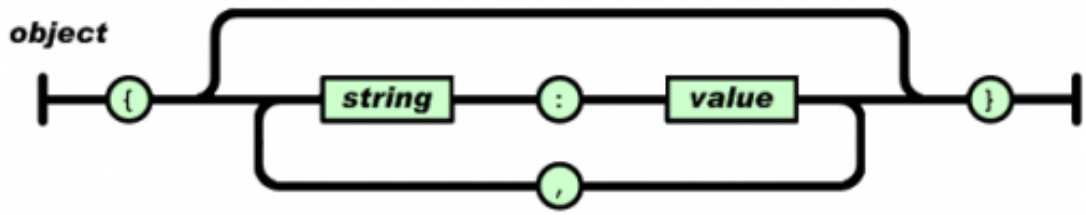

Figura 2 – Estrutura JSON

JSON é utilizado como protocolo padrão para os *Web services* e APIs existentes. Através de um arquivo JSON é possível dois sistemas totalmente diferentes se comunicarem entre si.

Por exemplo, um aplicativo deve enviar o nome do paciente e do médico para o servidor agendar uma consulta. Utilizando JSON a mensagem do aplicativo pro servidor seria da seguinte forma:

```
{
  "paciente": "José",
  "medico": "Ricardo"
}
```

O servidor ao processar entende que existe um objeto que possui dois atributos: paciente e médico, referindo-se ao nome de cada um. Depois de processar este objeto, o servidor deve enviar ao aplicativo o dia e a hora da consulta e se foi agendada com sucesso. Assim o servidor retorna o seguinte JSON para o aplicativo:

```
{
  "dat": "2017-09-17T13:00:00",
  "sucesso": true
}
```

O aplicativo recebe e processa o JSON entendendo que a consulta foi agendada com sucesso no dia 17 de setembro às 13h. O servidor e o aplicativo são sistemas diferentes, em plataformas diferentes, utilizando linguagens diferentes, porém conseguem transferir informações de um para o outro utilizando JSON.

2.10 GPS

Global Positioning System (Sistema de Posicionamento Global) é um sistema capaz de informar o posicionamento global (latitude, longitude e altitude) de um aparelho móvel em qualquer lugar da Terra sobre qualquer condição. Essa informação advém da interação de 3 satélites (ou mais para maior precisão) que trabalham com envio de ondas eletromagnéticas (PERES, 2016). Através do tempo de resposta do aparelho com essas ondas dos satélites, eles conseguem resolver sistemas de equações que determinam sua coordenada espacial.

Como as ondas utilizadas possuem a velocidade da luz, a informação de localização passa a ser, para métodos práticos, quase instantânea, com um erro de no máximo algumas dezenas de metros. É utilizado nas companhias de aviação e marítima para rastreamento, mas pode facilmente ter usos particulares como em veículos automotivos.

3 Materiais

Este capítulo descreve os materiais utilizados no projeto. Os materiais utilizados foram separados em duas categorias: servidor e plataforma móvel.

3.1 Servidor

Os materiais utilizados no servidor referem-se ao conjunto *software-hardware* utilizados para hospedar um site e as APIs utilizadas.

3.1.1 Raspberry Pi

Utilizado em muitos projetos acadêmicos devido a ampla funcionalidade e baixo custo, Raspberry Pi é um SBC (*Single Board Computer*). Ela realiza as atividades de um computador, porém seus componentes estão em uma única placa de circuito impresso.

Desenvolvida no Reino Unido pela Raspberry Pi Foundation, suporta os sistema operacionais baseados em GNU/Linux e Windows 10 IOT. Os modelos mais recentes da placa possuem melhor performance.

O modelo utilizado neste projeto foi a Raspberry Pi B+.

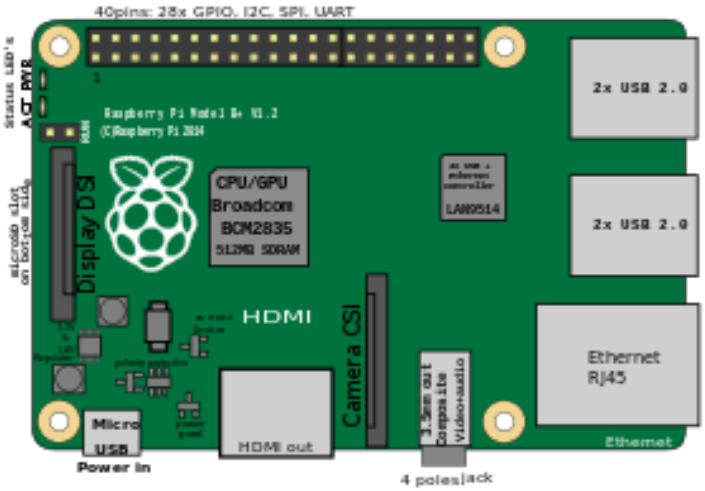

Figura 3 – Modelo utilizado: Raspberry Pi B+

Alguns dos recursos presentes neste modelo são:

- 4 portas USB 2.0
- Entrada para cartão microSD
- Saída HDMI
- Conector *Ethernet*
- Processador Broadcom SoC de 700MHz
- 512 MB de memória RAM

Este modelo foi escolhido devido a possibilidade de usá-la como servidor, além do excelente custo benefício.

3.1.2 Raspbian

Raspbian é uma variante do sistema operacional Debian (baseado em GNU/Linux) otimizada para RaspberryPi. Inclui as principais funcionalidades do sistema Debian e mais de 35000 pacotes pré-compilados para fácil instalação na Raspberry Pi (RASPBIAN, 2017).

A versão do sistema é a Raspbian GNU/Linux 8 (jessie). O sistema foi escolhido devido as funcionalidades, velocidade e estabilidade para o computador escolhido.

3.1.3 Python

Python é uma linguagem de programação de alto nível, funcional, interpretada e orientada a objetos. Possui recursos poderosos, um vasto conjunto de bibliotecas mantido pela comunidade e *frameworks* de terceiros.

A escolha da linguagem foi resultado do *framework* escolhido.

3.1.4 Django

Django é um *framework* Web baseado em Python que utiliza o modelo MTV(*model-template-view*). Este *framework* se baseia no principio DRY (*don't repeat yourself*) que faz com que o desenvolvedor aproveite ao máximo o código já feito, evitando a repetição.

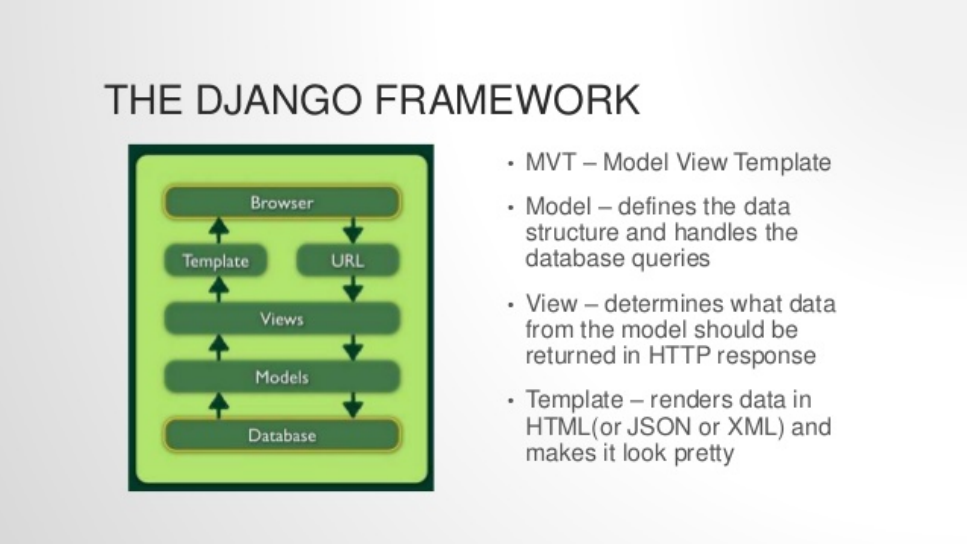

Figura 4 – Django Framework: estrutura MTV (SLIDESHARE, 2016)

Em Django (*Model*) define a estrutura dos dados trabalhados e lida com as requisições do banco de dados. Isto permite ao programador manipular os dados sem precisar escrever complexos códigos em SQL (DJANGOBOOK, 2017). *Views* determinam que dados do *Model* devem ser enviados para a resposta HTTP. E o *Template* renderiza de uma forma agradável em HTML a resposta.

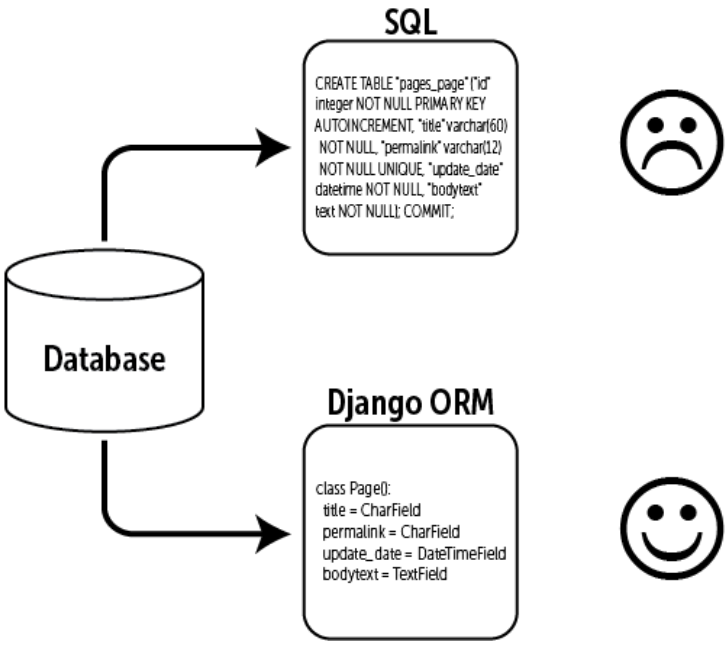

Figura 5 – Django ORM: Manipule dados sem escrever complexos códigos SQL Fonte:(DJANGOBOOK, 2017)

Este *framework* permite desenvolver de forma rápida um site com validações de formulário, manipulação do banco de dados e autenticações de forma simples e rápida.

Inicialmente foi escolhido o *framework* Rails, baseado na linguagem Ruby. Porém a baixa velocidade de execução do Rails em um hardware limitado limitou seu uso. Por isso optou-se pelo Django devido a velocidade de execução mesmo em *hardwares* mais simples, além do amplo suporte da comunidade e das diversas funcionalidades do *framework*.

3.1.5 Django Rest

O Django Rest Framework viabiliza de forma simples a criação de APIs REST em projetos Django. Esse *framework* agiliza o desenvolvimento de CRUDs (*Create Read Update Destroy*) e aprimora alguma das funcionalidades do Django.

3.1.6 FCM Django

O FCM é uma biblioteca Django que integra o Firebase FCM e o Django. Esta biblioteca permite enviar com uma linha de código a mensagem PUSH para um determinado dispositivo.

3.1.7 Firebase

Firebase é um conjunto de serviços fornecidos pela Google. Definido como BaaS(*Backend-as-a-service*) é um serviço de computação em nuvem que serve como *middleware*. Fornece aos desenvolvedores uma forma para conectar suas aplicações móveis e Web a serviços na nuvem a partir de APIs e SDKs.

O serviço utilizado foi o Firebase Cloud Messaging (FCM), solução para envio de notificações entre plataformas confiável e sem custo.

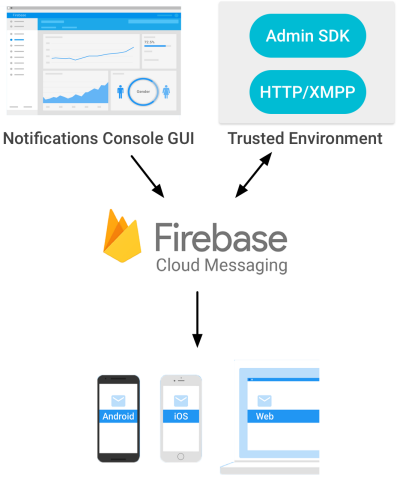

Figura 6 – Firebase Cloud Messaging

Este serviço permite que o servidor mande mensagens para os usuários do aplicativo desenvolvido através de notificações. Além disso, oferece um console para visualização e controle do serviço.

3.1.8 Google Maps APIs - Directions e Javascript

O conjunto de APIs do Google Maps, mantida pela Google, permite a visualização de mapas, cálculo de rotas, pesquisa por estabelecimentos dentre outros serviços. Pode ser utilizada em Android, iOS e Web.

Para o servidor foram utilizados o Google Maps Directions API e o Google Maps JavaScript API.

A API Google Maps Directions permite que um *request* composto de todas as informações da viagem (origem, destinos, tipo de transporte) resulte em um arquivo JSON com o melhor trajeto e detalhes como distância, tempo estimado, ordenação entre outros valores.

Figura 7 – Google Maps Directions API

A API Google Maps Javascript permite a visualização de um mapa acompanhado de marcadores ou de um conjunto de linhas que representa um trajeto. É composta de muitas ferramentas que utilizam o Javascript para interação, manipulação e visualização do Mapa.

Figura 8 – Mapa com marcador utilizando Google Maps Javascript API

3.1.9 MariaDB

MariaDB é um sistema de gerenciamento de banco de dados relacional desenvolvido pelo próprios desenvolvedores do MySQL após este ser adquirido pela Oracle. MariaDB é mantido atualizado com a última versão do MySQL e funcion exatamente como MySQL. Todos os comandos, interfaces, bibliotecas e APIs que existem no MySQL também existem no MariaDB.

O banco de dados em MariaDB foi escolhido devido ao melhor desempenho e segurança em relação ao MySQL.

3.2 Plataforma Móvel

3.2.1 Android

Android é um sistema operacional baseado em Linux que opera em *smartphones*, *netbooks* e *tablets*. Foi criado pela empresa Open Handset Alliance, uma aliança entre várias empresas, dentre elas a Google. Pesquisas apontam que 90% dos usuários de *smartphones* no Brasil usam aparelhos com sistema operacional Android (TECNOBLOG, 2016).

3.2.2 Android Studio

É um ambiente de desenvolvimento integrado criado especificamente para Android. Lançado em Maio de 2013 se encontra na versão 2.3.3. A figura abaixo mostra um pouco da interface e ferramentas do Android Studio.

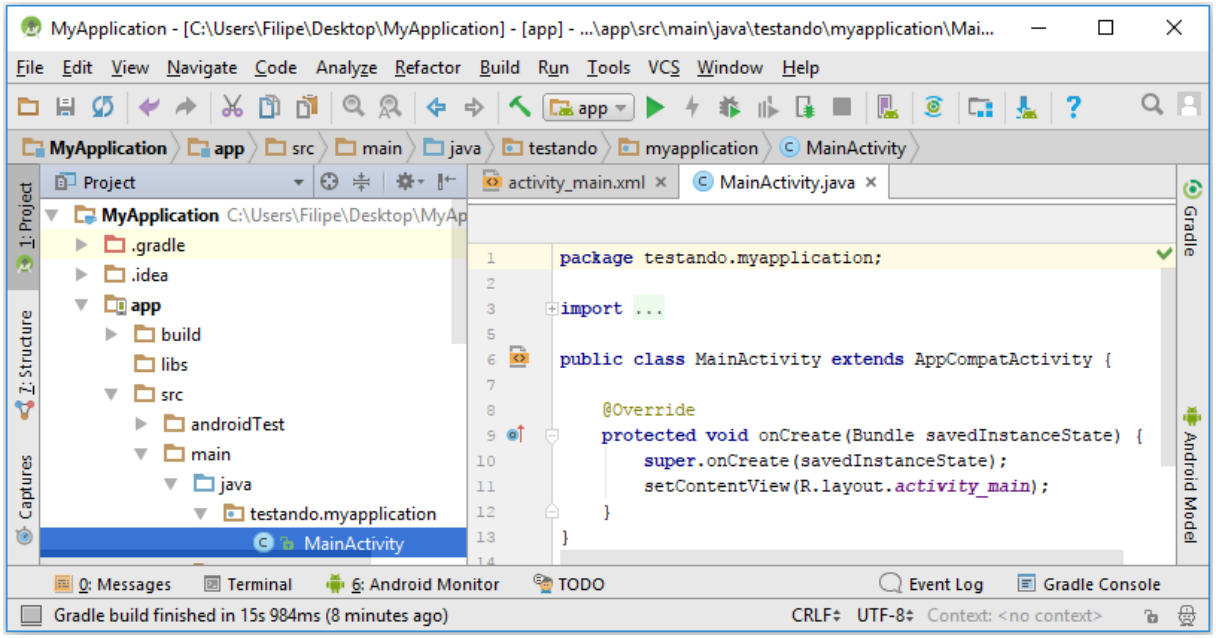

Figura 9 – Interface do Android Studio

Após escrever o código e clicar em "run app", pede-se o modelo do *smartphone* para a simulação. Além de processar o aplicativo, é possível manipulá-lo de diversas formas, como alterar configurações de Wifi, bateria, dados móveis, localização e navegar na Internet.

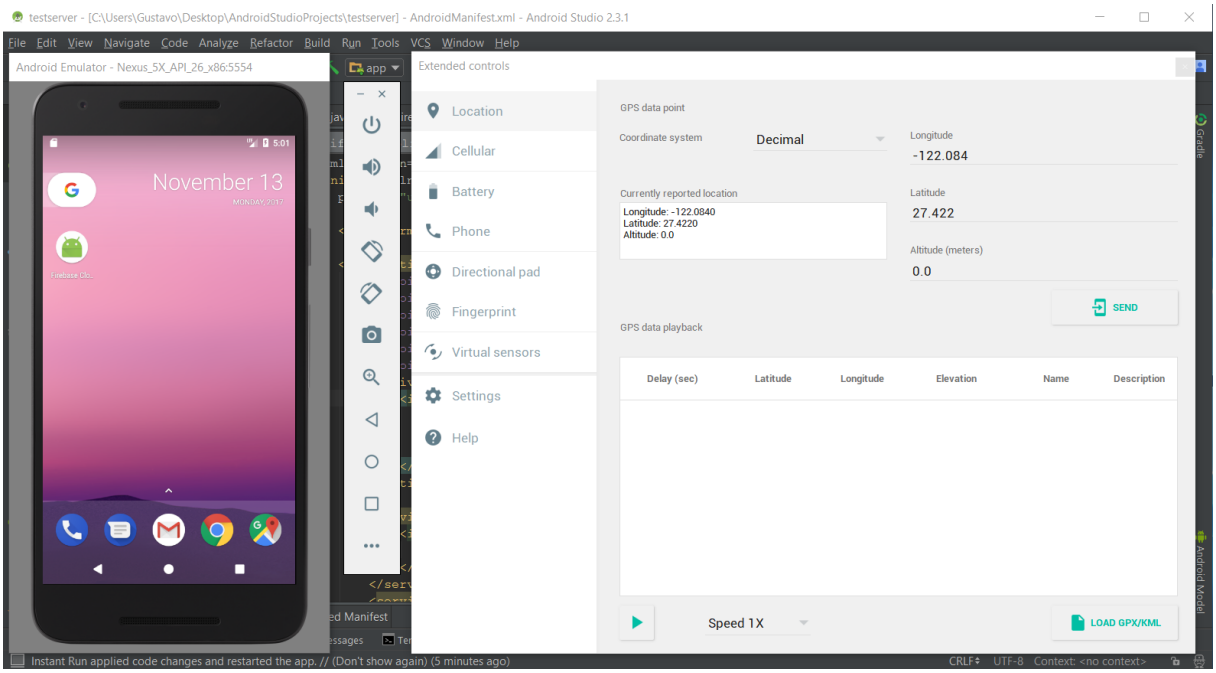

Figura 10 – *Smartphone* do simulador do Android-Studio

É possível simular o aplicativo a ser desenvolvido no modo *Debug*. Nele você pode interromper o processamento do código na linha desejada e obter valores das variáveis naquele local. É muito útil para entender o funcionamento do App e para corrigir eventuais problemas.

3.2.3 Java

Java é a linguagem de programação mais utilizada no mundo atualmente (ORACLE, 2017). É uma linguagem interpretada e orientada a objetos, possuindo uma variedade ampla de bibliotecas e APIs e possui uma sintaxe similar a C++, e é utilizada nos projetos de aplicativos do Android Studio.

3.2.4 ZXing

ZXing (advém de Zebra-Crossing) é uma biblioteca de código aberto para leitura de códigos de barra, tanto unidimensionais quanto bidimensionais. Ele é acompanhado de funções que o tornam bastante completo, como histórico de leitura, opção de compartilhamento e uma variedade de opções para a câmera. Permite escolher diversas ações ao se ler um código, como vibrar, apitar, copiar e abrir a página Web caso o código carregue uma URL.

3.2.5 Gson

Gson é uma biblioteca de código aberto que auxilia na conversão de objetos ou lista de objetos em JSON e vice-versa. Foi criado pela Google em 22 de Maio de 2008 e sofre constantes alterações, já se encontrando na versão 2.8.2. Abaixo um exemplo do seu uso. Primeiramente um objeto "game1" é declarado, atribui-se valores e em seguida utiliza-se o método "toJson" da biblioteca Gson para convertê-lo em JSON:

```
Game game1 = new Game();
game1.nome = "Super Mario Bros";
game1.desenvolvedor = "Nintendo";
game1.genero = "Plataforma";

Gson gson = new Gson();
String json = gson.toJson(game1);
```

A saída para a variável "json" é:

```
{
  "desenvolvedor": "Nintendo",
```

```
    "genero": "Plataforma",
    "nome": "Super Mario Bros"
}
```

Através do método "fromJson" é possível realizar o processo inverso. As entradas são um JSON (no exemplo seria a String acima) e a classe do objeto ("Game"). O método retorna o objeto gerado ("game1").

3.2.6 OkHttp

É uma biblioteca muito eficiente para se fazer requisições HTML. Através dos seus métodos, o código que conecta aplicativo e servidor torna-se mais simples e menos verbosa. Um exemplo de código para fazer um GET à um servidor qualquer:

```
OkHttpClient client = new OkHttpClient();
Request request = new Request.Builder()
    .url("URL_DO_SERVIDOR")
    .get()
    .build();
Response response = client.newCall(request).execute();
String resposta = response.body().string();
```

A variável "resposta" contém, agora, o JSON de resposta ao GET para o URL dado.

3.2.7 Google Maps API - Android

No aplicativo o API é utilizado para mostrar um determinado local do mapa. O código abaixo é responsável por abrir o Google Maps centrado nas coordenadas (23.5453221,-46.4748148):

```
Uri gmmIntentUri = Uri.parse("geo:-23.5453221,-46.4748148");
Intent mapIntent = new Intent(Intent.ACTION_VIEW, gmmIntentUri);
mapIntent.setPackage("com.google.android.apps.maps");
startActivity(mapIntent);
```

4 Métodos

Este capítulo aborda como foi o planejamento do projeto, sua estrutura, organização e execução.

4.1 Planejamento

Para correto entendimento foi necessário definir as seguintes expressões:

- Motorista, entregador: funcionário da transportadora que realiza a entrega de encomendas;
- Transportadora, distribuidora: empresa que gerencia motoristas através de um sistema logístico para entregar de encomendas;
- Cliente: recebe as encomendas da transportadora;
- Encomenda, produto, mercadoria: qualquer tipo de produto enviado pela transportadora.

O sistema faz a integração entre as seguintes partes: motorista, transportadora, cliente. O motorista deve enviar suas coordenadas para a transportadora além de outras informações discutidas adiante. A transportadora deve colocar as informações das encomendas no sistema, receber informações de seus motoristas e enviar informações ao cliente. O cliente deve receber informações de sua encomenda.

Aplicando em um caso prático: quando um usuário realiza uma compra online a empresa que comercializa o produto contrata o serviço de uma empresa transportadora. A transportadora recebe da loja o produto e alguma de suas informações essenciais para entrega, como o endereço de entrega, nome do cliente. A transportadora providencia o transporte do produto através de motoristas para que este seja entregue em segurança e o mais rápido possível.

Com os dados recebidos é possível determinar através de logística como esse produto deve ser transportado e junto com quais outros produtos. Esta questão não foi abordada no projeto, já que existem diversas soluções para esta questão no mercado.

A questão abordada é sobre como será realizada a distribuição e em que ordem será feita no dia da entrega. Com os endereços dos destinatários é possível calcular o melhor trajeto para cobrir todas as encomendas, definindo a ordem de entrega e reduzindo os gastos com recursos da empresa. Sabendo a ordem de entrega é possível determinar uma

estimativa de que hora o produto vai ser entregue para o cliente. Assim o cliente pode fazer seus preparativos para estar em casa no horário estimado pela transportadora.

Este caso prático é mostrado na figura 11. A ordem em que todo o ciclo é feita está descrita nos números indicadores. Os indicadores em cinza mostram o ciclo da transportadora (motoristas e funcionários) e o servidor. E os indicadores em verde mostram o ciclo do cliente-servidor.

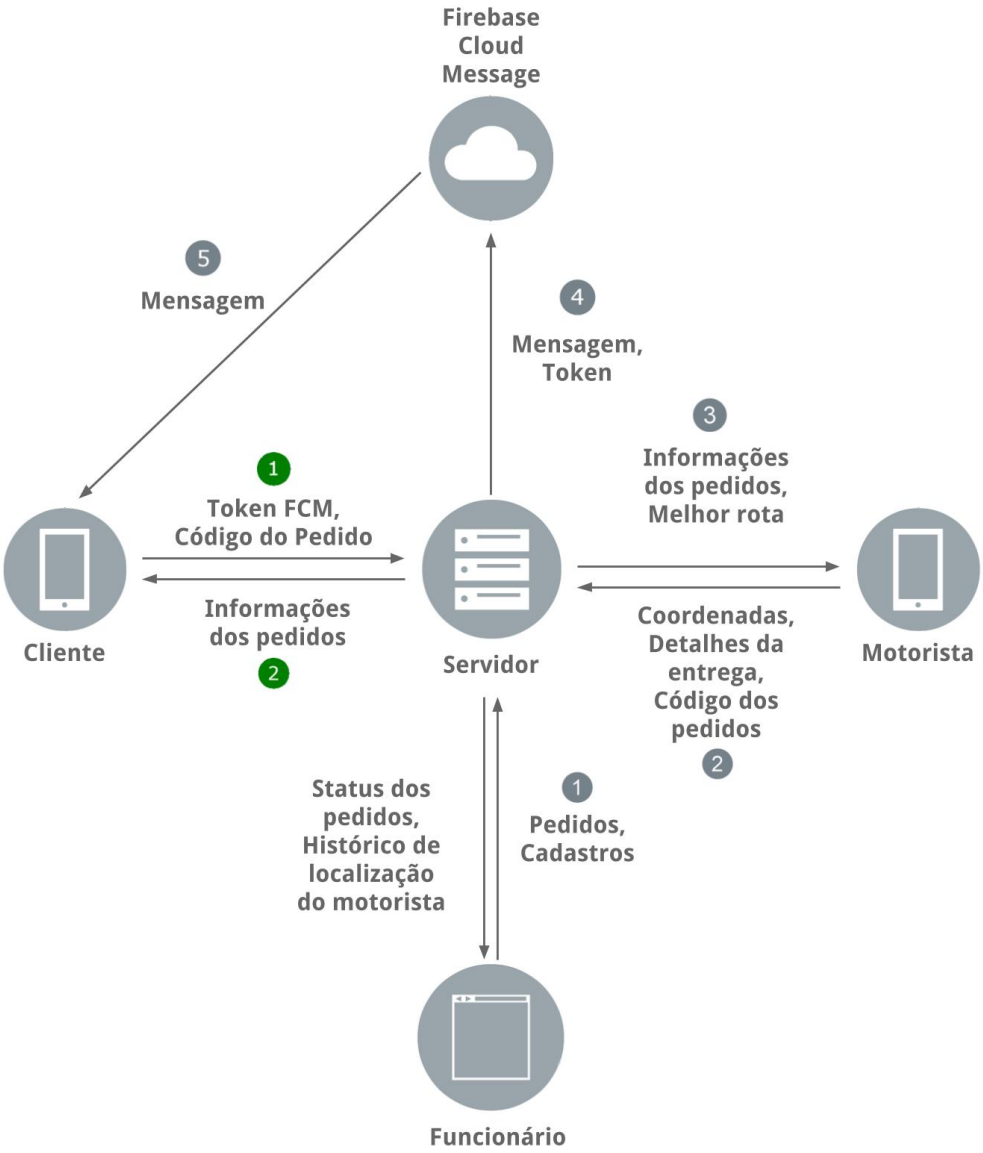

Figura 11 – Arquitetura geral do projeto

Primeiro um funcionário cadastra o pedido e etiqueta a encomenda (1c - cinza). Quando for carregar o veículo com as encomendas o motorista lê a etiqueta com o código QR de cada encomenda e faz a requisição para o servidor com os códigos das encomendas (2c). O servidor retorna as informações do pedido, como endereço, nome do destinatário

e a ordem das entregas para melhor rota (3c). A seguir o servidor envia a mensagem para o serviço FCM do Google com as informações da entrega e o *token* do dispositivo destinatário (4c). O FCM enviará a mensagem para o cliente específico através de uma notificação PUSH (5c). Estas são as ações realizadas pelo sistema e pela transportadora.

Quando o cliente logar no aplicativo irá cadastrar um código da encomenda. O aplicativo enviará o código da encomenda junto com o *token* específico do dispositivo para o servidor (1v - verde). O servidor irá associar o *token* à encomenda e quando tiver atualizações sobre a entrega ocorrem os passos (4c) e (5c). Se o cliente tiver cadastrado os códigos das encomendas, quando ele entrar no aplicativo receberá as informações atualizadas dos pedidos (2v).

Com estas informações foi realizado o planejamento do projeto. Primeiro foi necessário separar o que cada uma das partes irá utilizar. Para o motorista um sistema móvel instalado em um *smartphone* ou *tablet*. Para a transportadora um sistema Web que permite visualizar e processar os dados recebido dos motoristas e das encomendas. E para o cliente um site em que inserindo o código de rastreio ele consiga ver a estimativa de entrega, junto com um aplicativo que informe através de notificações alertando o usuário da entrega do produto.

Baseado nestas duas informações separou-se o projeto em duas partes: servidor e aplicativo.

4.2 Servidor

Ao analisar os requisitos estruturou-se o site conforme o diagrama da figura 12.

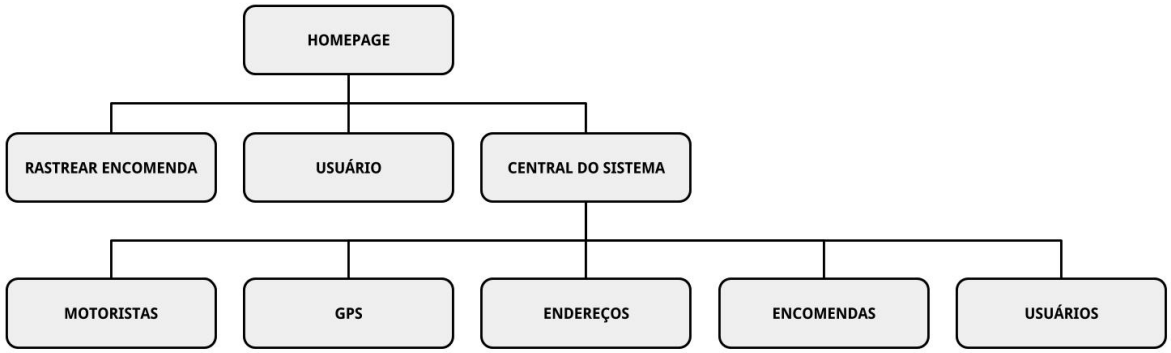

Figura 12 – Estrutura do Site

A principal página (*index*) é onde a empresa transportadora mostra suas informações e serviço. Também há um formulário que redireciona para a página rastrear encomenda.

Estas páginas são abertas e não exigem autenticação. A página usuário permite o cadastro e login de cliente. Ao logar-se mostra-se o histórico de encomendas entregues e a serem realizadas.

Com base nas relações para projetar o banco de dados foram definidos os seguintes modelos: *Coordinate*, *Address*, *Order*, *FCM*, *Profile*, *User*, *Group*. Com base nestes modelos foi projetado o banco de dados segundo diagrama da Figura 13.

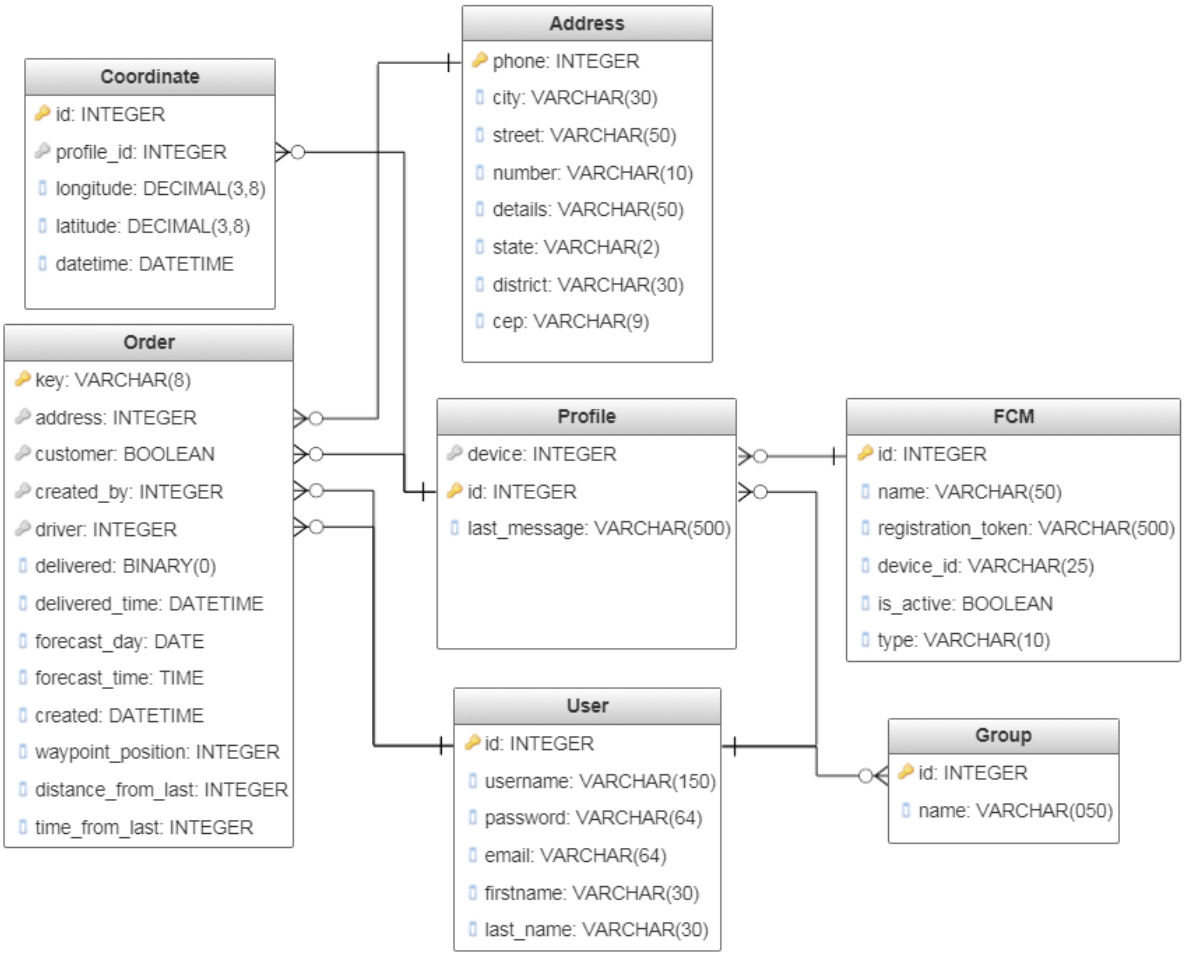

Figura 13 – Diagrama do Banco de Dados

Dentro da central do sistema há três tipos de usuário: motoristas, funcionários, e administradores. Estes tipos são definidos pelos grupos associados a cada User. As permissões de cada tipo estão na tabela 3. Clientes não podem acessar a central do sistema.

Tipo de Usuário	Profiles	GPS	Endereços	Encomendas	Usuários
Administrador	C R U D	C R U D	C R U D	C R U D	C R U D
Motorista		C		R U	
Funcionário			C R U D	C R U	
Legenda: Create: criar, **R**ead: ler, **U**pdate: modificar, **D**estroy: apagar					

Tabela 3 – Permissões para os tipos de usuário

Para registrar um novo endereço no sistema é necessário acessar a página de cadastros de endereço. Ao inserir o CEP o formulário já preenche automaticamente os outros campos (rua, bairro, cidade) com base no CEP inserido. O cadastro de endereço é feito pelo formulário representado na figura 14.

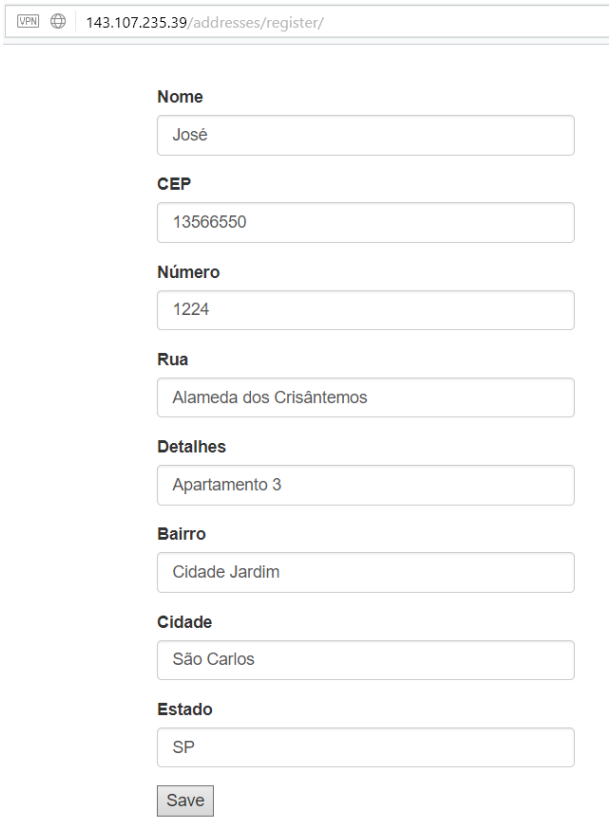

Figura 14 – Formulário de Cadastro de Endereço

Dentro do sistema há como ver o percurso realizado pelos motoristas. Ao acessar a página de registros existe um formulário que permite selecionar o motorista e o dia. Ao preencher o formulário o site retorna o caminho realizado pelo motorista, conforme observa-se na figura 15.

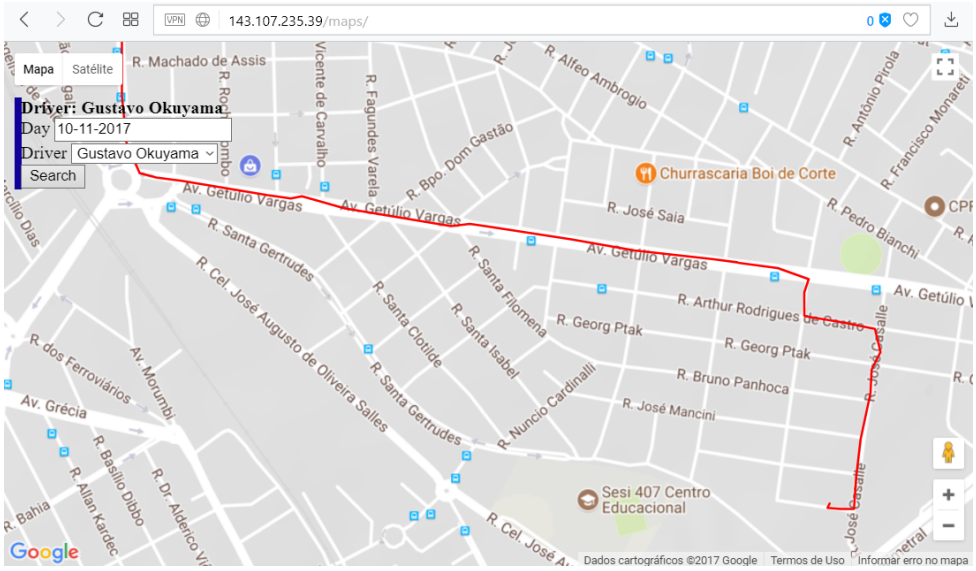

Figura 15 – Histórico do percurso realizado pelo motorista

Todas as outras páginas não tem *layout* personalizado. Elas aproveitam o layout gerado pelo *framework* Django REST. Nelas é possível acessar, criar, editar e excluir cada elemento. Essas páginas são utilizadas apenas para testes. Elas concentram todas as funções em uma única página. Uma versão do site para produção deve substituir essas páginas por *templates* personalizados. Um exemplo de página gerada pelo Django REST é mostrado na figura 16.

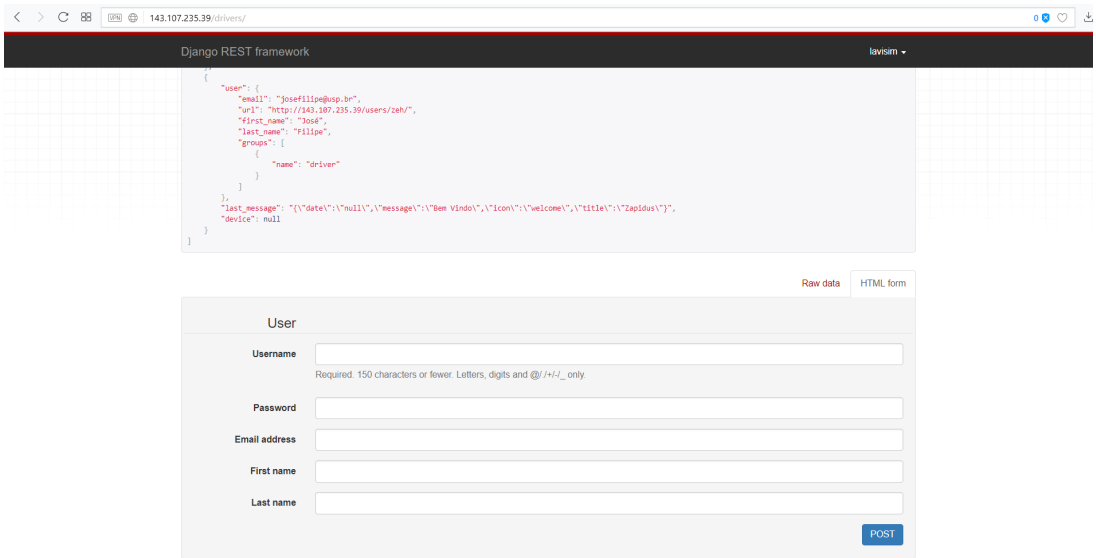

Figura 16 – Template do Django REST para recurso Drivers

4.3 Aplicativo do Cliente

As telas iniciais (telas de login) dos aplicativos podem ser representadas pelo fluxograma da figura 17.

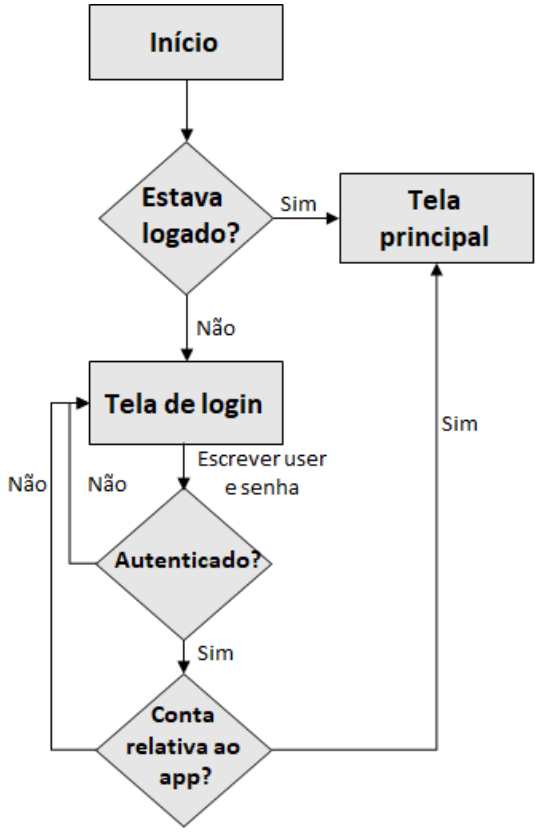

Figura 17 – Fluxograma da tela de login

A figura 18 é uma *screenshot* da tela inicial retirada diretamente do simulador do Android Studio. Foi colocado propositalmente uma senha inválida. O Aplicativo informa, além de conta não registrada, se houve problemas de conexão com o servidor ou se a conta está sendo utilizada no aplicativo incorreto (motoristas só são autenticados no app do motorista e cliente no app de clientes).

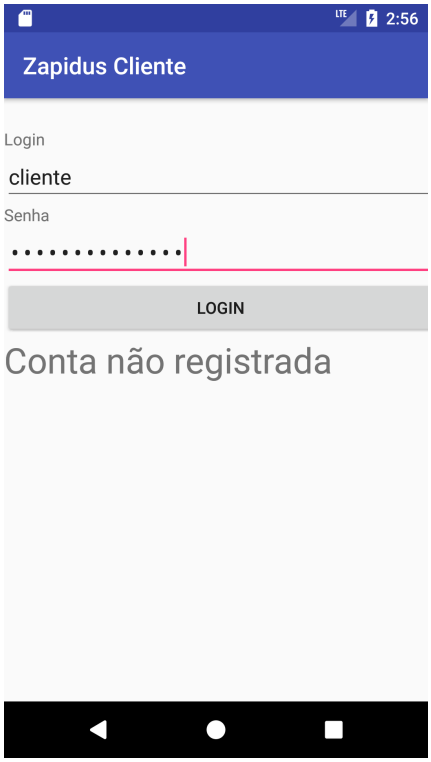

Figura 18 – Screenshot da Tela de login

Feito login, o aplicativo vai para a tela principal. Nela o usuário pode ler o código QR recebido da empresa transportadora a fim de obter informações do seu produto. Caso ele queira ter informações de várias encomendas, ele pode ler vários QRs (ou até adicionar o código manualmente) criando uma lista. Ao se clicar em algum elemento da lista, algumas informações são dadas junto com alguns botões. Eles são:

- Atualizar: as informações sobre a encomenda são atualizadas. Por exemplo, a localização do motorista.
- Localizar: caso a encomenda esteja vinculada à algum motorista, é mostrado através do aplicativo Google Maps a sua última localização enviada para o servidor. Caso não esteja, um diálogo informa que não há vínculo.
- Deletar: aquele elemento é deletado.

A figura 19 mostra uma *Screenshot* da tela principal.

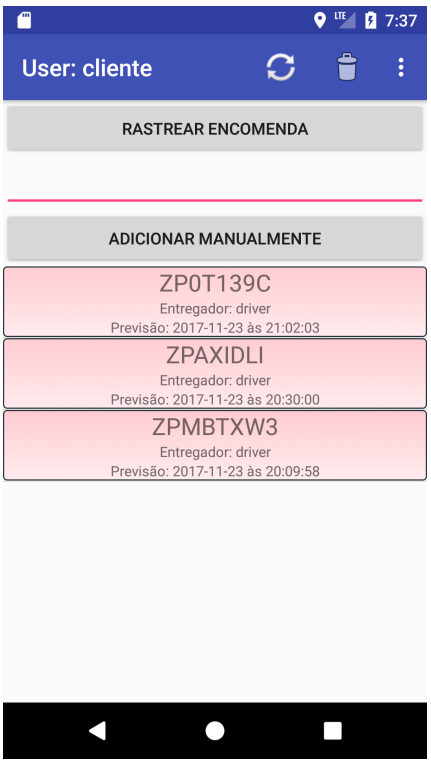

Figura 19 – Screenshot da tela principal do aplicativo do cliente

Os elementos da lista podem estar em 3 cores distintas:

- Vermelho: o produto ainda não foi entregue.
- Amarelo: o transportador levou o produto até o endereço, mas não houve quem o pudesse receber. Haverá o retorno no dia seguinte. Caso o produto não seja recebido por 3 vezes, haverá um aviso para que o produto seja buscado no endereço especificado.
- Verde: o produto já foi entregue ao destinatário.

A fim de informar o usuário de novas alterações nas encomendas, o servidor providencia notificações PUSH. Nas opções, presentes na tela principal, consta um histórico dessas notificações. Tanto esse histórico como todas as informações das encomendas na lista são armazenados na memória interna do dispositivo.

O fluxograma da figura 20 explica o funcionamento geral da tela principal.

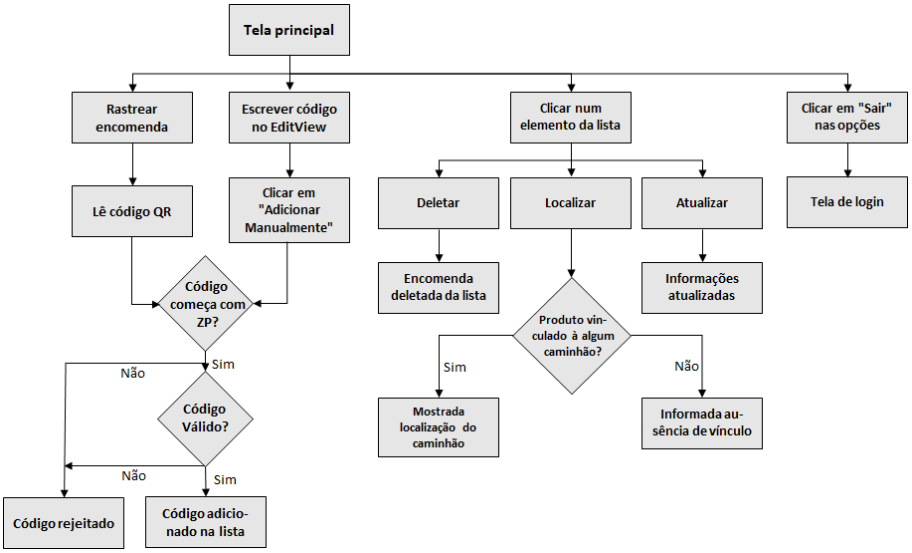

Figura 20 – Fluxograma da tela principal do App do cliente

4.4 Aplicativo do Entregador

A tela inicial (login) para o aplicativo do Entregador segue a mesma lógica do aplicativo do cliente. O que os diferencia é apenas a separação, no servidor, entre contas de entregadores e de clientes. A parte de leitura de códigos QR e adicionar encomendas manualmente também existem, da mesma forma, nesse App. As diferenças ocorrem, principalmente, nas informações que a lista fornece de cada produto, na caixa de diálogo ao se clicar em algum elemento da lista e na presença do rastreador. A figura 21 mostra a tela principal do aplicativo do entregador.

Figura 21 – Screenshot do aplicativo do Entregador

Nota-se, no topo da tela, um botão referente ao rastreador. Ele muda de cor ao

ser clicado alternando entre verde (ligado) e vermelho (desligado). O fluxograma do seu funcionamento está representado na figura 22.

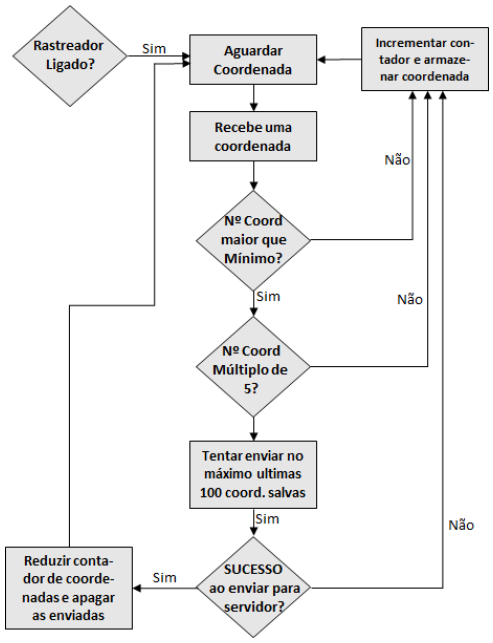

Figura 22 – Fluxograma GPS

Fez-se necessário enviar para o servidor apenas números múltiplos de 5 de coordenadas simultaneamente para diminuir a quantidade de chamadas ao mesmo. Se um grande número de motoristas rastrearem ao mesmo tempo, exigir demais do servidor pode gerar problemas. O número mínimo de coordenadas a ser enviada de uma vez pode ser escolhido pelo motorista clicando nas opções e "Configurações". Ele encontrará a tela da figura 23.

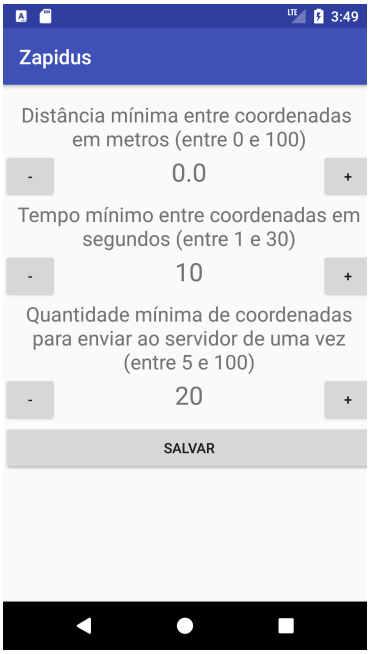

Figura 23 – Configurações GPS

Aqui ele pode também alterar as configurações de GPS, como o tempo mínimo e distância mínima entre duas coordenadas consecutivas.

A respeito da lista, o que diferencia os dois aplicativos é o diálogo imediato ao clique. Nesse caso tem as seguintes ações:

- Cheguei ao local: Deve ser pressionado quando o motorista chegou ao local daquela encomenda e tentou contato para a entrega. Ao clicar nessa opção, outro diálogo é aberto questionando se o produto foi ou não entregue. É importante ressaltar que clicar em não, nesse caso, não é o mesmo que cancelar: ele está dizendo ao servidor que o receptor do produto não se encontrava no endereço. É assim que a cor da lista passa a ser amarela para o cliente. Clicando em "sim" informa-se o sucesso no envio do produto.

- Atualizar: atualiza as informações daquela encomenda. Por exemplo, alterações nos detalhes do produto.

- Opções: abre outra caixa de diálogo com duas opções: mostrar o respectivo endereço no Google Maps e a opção de deletar aquele elemento da lista.

Todos as encomendas registradas e as coordenadas GPS (que ainda não foram enviadas) são armazenadas na memória interna do dispositivo. Portanto, não há risco de perder esses dados ao fechar o aplicativo.

Por fim, entre as opções existe o "Cálculo de Melhor Trajeto". Ao selecionar essa opção, é requisitado ao servidor que ele reúna todos os endereços cadastrados na respectiva conta e através do Google Maps API consiga o melhor trajeto possível para redução de custo. Existe uma deficiência nesse método do Google pois ele recebe no máximo 20 endereços para se otimizar trajeto e existe um limite de requisições diárias para isso.

4.5 Integração entre Servidor e Aplicativo

Existe integração entre ambos nos seguintes momentos:

- Token de login: ao se efetuar corretamente o login (em ambos os Apps) o servidor retorna ao aplicativo um Token (uma String extensa de dígitos alfanuméricos aleatórios). O aplicativo armazena esse Token na sua memória para que, posteriormente, mantenha-se conectado na sua conta mesmo depois de fechar o App. Esse token também é necessário para se comunicar com o servidor para os demais momentos abaixo.

- Login automático: quando o usuário abre o App novamente, porém anteriormente já estava conectado à sua conta, o servidor verifica se o Token armazenado confere com o do banco de dados e em caso positivo o usuário é redirecionado automaticamente para a tela principal.
- Requisição de informações de uma encomenda pelo seu código: ambos os aplicativos exigem informações registradas no banco de dados a respeito de uma encomenda ao fornecê-lo seu código, seja pela leitura com a câmera ou envio manual.
- Notificações PUSH: o servidor envia, através do FCM, notificações PUSH para o aplicativo do cliente a respeito de atualizações nas suas encomendas.
- Atualizar dados: ambos Apps pedem atualizações de dados de encomendas para o servidor.
- Envio de Coordenadas GPS: O aplicativo do motorista envia coordenadas GPS para o servidor, que ficam armazenadas para monitoramento dos administradores e dos clientes.
- Melhor Trajeto: O App do motorista faz uma requisição ao servidor de melhor trajeto de todos os endereços por ele cadastrados. O servidor retorna os mesmos endereços na ordem otimizada.

5 Resultados e Discussões

5.1 Aplicativo do Motorista

Nessa seção serão analisados os resultados do aplicativo em um smartphone Moto G3 e em seguida será feita uma conclusão baseado no que se esperava.

5.1.1 Rastreador

O rastreador funcionou conforme esperado. Diversos testes foram feitos atribuindo diversas configurações diferentes, e em nenhum foi encontrado irregularidade. O tempo de envio das coordenadas ao servidor foi suficientemente baixo. Por padrão são enviadas de 10 em 10, e nesse caso o tempo de envio foi abaixo de meio segundo.

Foram feitos testes para quando não há conexões entre celular e servidor (ausência de Wi-Fi, dados móveis ou queda de servidor). Esperava-se que o dispositivo armazenasse coordenadas até que a conexão fosse restabelecida e o envio fosse feito em pacotes de no máximo 100 coordenadas. Funcionou corretamente.

Também foi analisado o seu comportamento com respeito ao armazenamento das coordenadas na memória interna. Ao fechar o aplicativo e abri-lo novamente, as coordenadas que não haviam sido enviadas foram recuperadas até que fossem enviadas, assim como planejado.

5.1.2 Adição de novas encomendas via QR Code ou teclado

A adição de encomendas ao aplicativo funcionou corretamente. O app foi capaz de manipular corretamente o código para evitar erros como formato, inexistência no banco de dados e repetição.

5.1.3 Entrega e não entrega de encomendas

Quando uma encomenda é recebida, o motorista deve selecionar, no aplicativo, o respectivo item da lista referente aquele endereço e selecionar que o produto foi recebido. Foi feito o teste, e o item da lista mudou da cor vermelha para verde, como esperado. O teste também foi feito para o não recebimento, e o item ficou amarelo.

Foi verificado no banco de dados que as informações são atualizadas corretamente após a postagem.

5.1.4 Requisição de melhor rota

Dentre as opções do menu no ActionBar do aplicativo existe a opção de cálculo de melhor rota. Ele pede para que o servidor gere uma rota otimizada para a realização de todas as entregas através do Google Maps API. Como isso envolve comunicações entre as 3 partes, o processo acaba sendo demorado, além de ser limitado a no máximo 20 encomendas.

Foram feitos diversos testes. Não funcionou completamente como esperado, mas o resultado foi aceitável. Primeiro, leva em torno de 10 segundos, considerado um tempo longo. Segundo, às vezes ocorre algum erro aleatório que força o usuário tentar realizar o processo novamente.

Foi feito um teste para averiguação da rota. Gerou-se 10 endereços aleatórios na cidade de São Carlos e foi feita a requisição de melhor trajeto. Visualmente, na tela do celular, ocorreu apenas uma mudança na ordem dos elementos da lista, estando agora otimizada para menor percurso. A figura 24 ilustra essa variação.

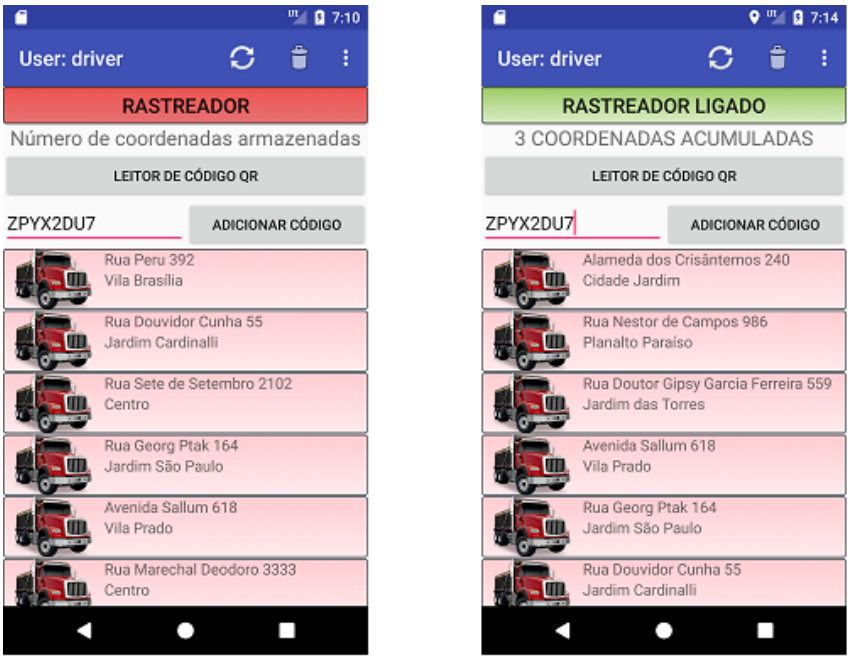

Figura 24 – Antes e depois do cálculo de melhor rota

Para verificação, foi gerada a rota passando pelos 10 endereços no Google Maps, e obteve-se o resultado da figura 25.

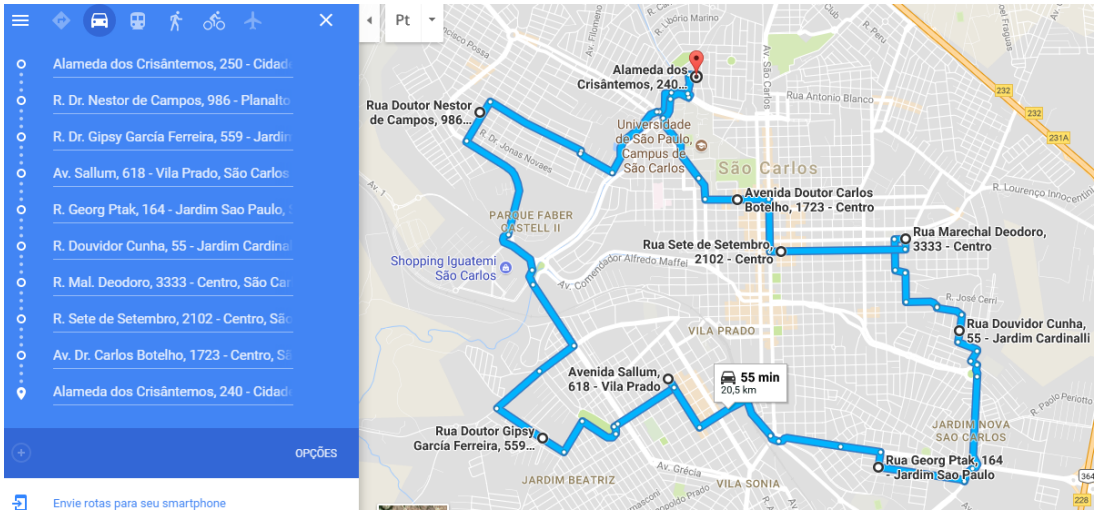

Figura 25 – Rota gerada pelo recurso do Google Maps API

Pode-se observar que a rota gerada passa por todos os endereços realizando um percurso visualmente bom, sem passar por locais desnecessários ou realizando caminhos claramente incorretos.

5.1.5 Gasto de bateria

O teste de gasto de bateria foi realizado mantendo o aplicativo em funcionamento por 5 horas consecutivas. O rastreador estava ligado e recebendo coordenadas a cada 10 segundos. Os dados móveis estavam ligados para o envio das coordenadas e nenhum outro aplicativo funcionava durante o teste. A figura 26 nos mostra o resultado:

Figura 26 – Bateria utilizado pelo app do motorista em 5 horas

O aplicativo utilizou apenas 1% da bateria em 5 horas, mostrando-se bem econômico nesse sentido.

5.1.6 Uso de memória

Foram mantidas as mesmas condições do teste de bateria. A imagem 27 mostra o uso de memória máximo de ambos os apps.

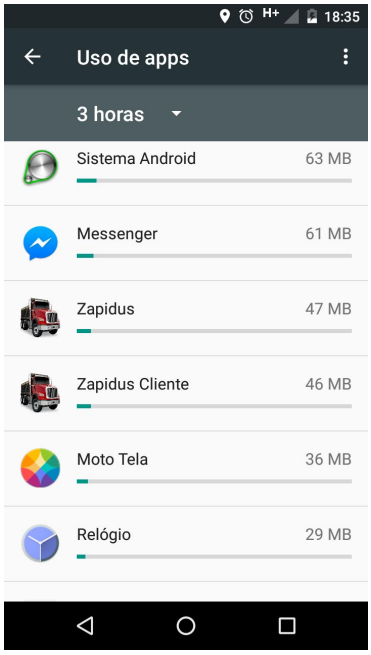

Figura 27 – Memória máxima utilizada pelos apps

Nota-se que ambos os aplicativos não chegaram a usar 50Mb, um valor baixo.

5.1.7 Armazenamento

O aplicativo, com a lista de endereços utilizada no cálculo de rota, ocupa um espaço de 2,94 MB do dispositivo, sendo 2,8 MB do programa e 140 KB de dados. Isso é uma quantia baixa em comparação com outros aplicativos populares.

5.2 Aplicativo do Cliente

Os testes com o aplicativo do cliente foram feitos no mesmo *smartphone* do do aplicativo do entregador.

5.2.1 Requisição de informações via QR Code ou teclado

Assim como no aplicativo do motorista, funcionou corretamente e foi verificado no banco de dados que houve correta vinculação entre código e cliente após a leitura.

5.2.2 Atualização das informações

A atualização das informações (tanto num único item da lista como em todos de uma vez através do botão do ActionBar) funcionaram como esperado.

5.2.3 Envio do *token* FCM

O *token* FCM deve ser enviado para o servidor no momento do login do cliente. Foi verificado no servidor seu funcionamento, inclusive verificado também que quando o logout é realizado, o *token* é removido.

5.2.4 Recebimento de notificações via FCM

Quando o servidor gera uma rota, é possível estimar o horário de recebimento de cada uma das suas respectivas encomendas. Assim, com esse dado em mãos, uma notificação com o horário estimado é enviada via Firebase para os clientes vinculados aos produtos.

Foi feito um teste, e a notificação chegou corretamente conforme a figura 28.

Figura 28 – Notificação ao cliente

Verificou-se que a notificação chega até mesmo quando o aplicativo está fechado. Ao clicar na notificação, nesse caso, o usuário é levado a tela principal do aplicativo.

5.2.5 Verificação da localização do motorista

No momento em que o motorista gera a rota otimizada e o cliente recebe a notificação do horário previsto de entrega, são vinculados no banco de dados motorista e cliente. A partir desse momento é possível verificar a última localização registrada pelo motorista clicando sobre o botão "localizar". Foi feito um teste e verificado seu funcionamento, como mostra a figura abaixo:

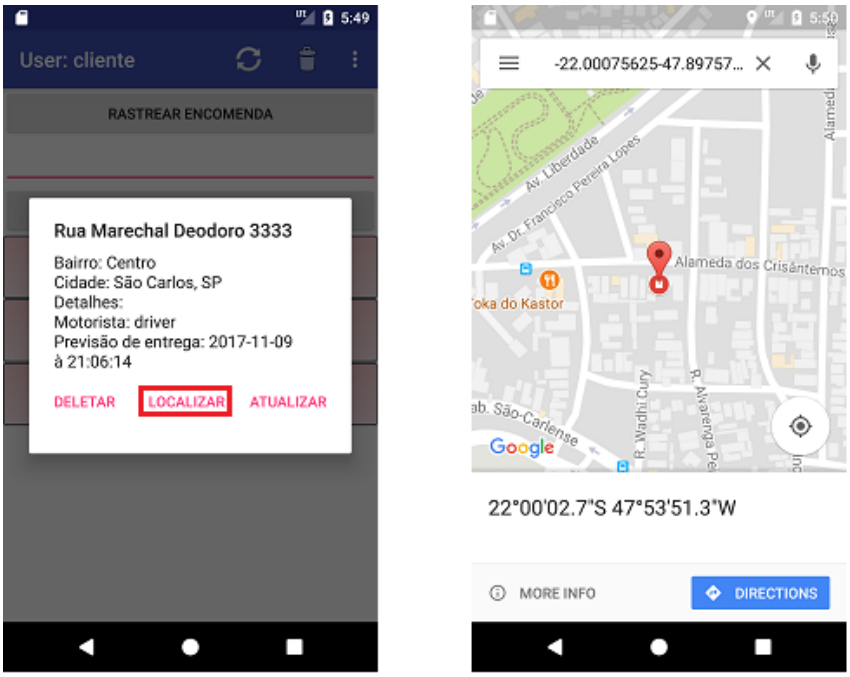

Figura 29 – Ferramenta de localização do aplicativo do cliente

5.2.6 Armazenamento

O aplicativo, com a lista de endereços utilizada no cálculo de rota, ocupa um espaço de 3,28 MB do dispositivo, sendo 3,09 MB do programa e 192 KB de dados. Assim como o outro app, é muito leve.

5.3 Servidor

5.3.1 Rastreamento do motorista

No banco de dados do servidor estão armazenadas todas as coordenadas de todos os motoristas usuários do aplicativo. Isso torna possível a verificação de todo o percurso feito por eles. Para isso foi criada uma ferramenta onde seleciona-se o motorista e o dia em que se quer obter informações. Foi feito um teste viajando-se entre as cidades de Pontal e Sertãozinho utilizando o rastreador como mostra a figura 30.

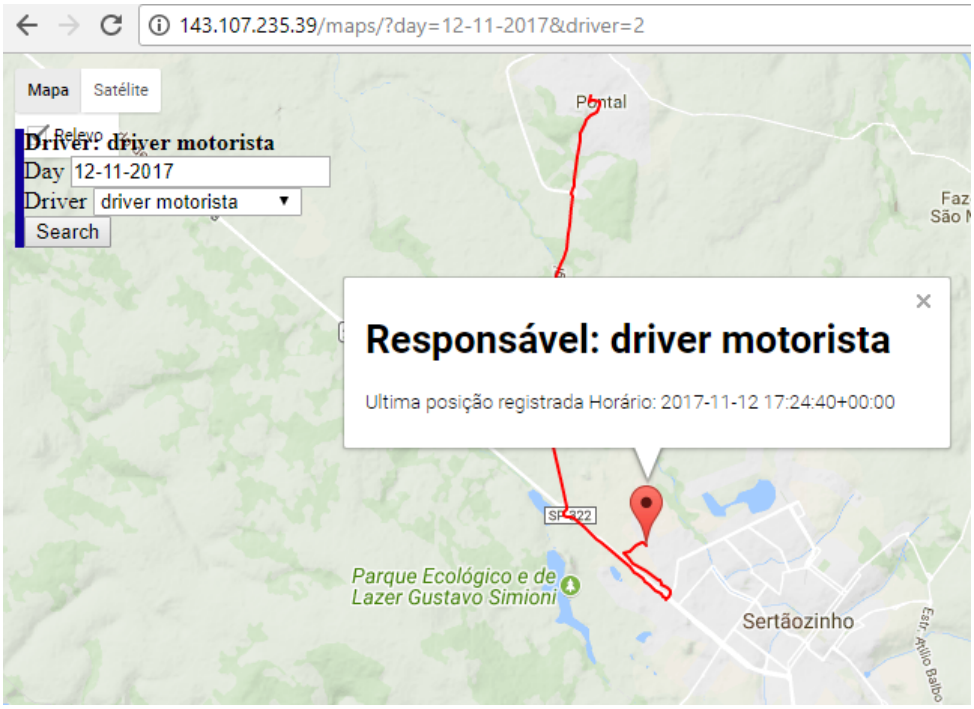

Figura 30 – Ferramenta de rastreio dos motoristas

O trajeto mostrado foi exatamente o realizado, assim como o horário de chegada notificado pela mensagem no balão.

5.3.2 Teste de memória e Processamento

Foi utilizado o comando "free -m -t" para obter informações gerais do uso de memória da Raspberry Pi com apenas o servidor em execução. A figura 31 mostra o resultado.

```
root@raspberrypi:~# free -m -t
             total       used       free     shared    buffers     cached
Mem:           434        353         80         12         54        140
-/+ buffers/cache:        158        275
Swap:          611          0        611
Total:        1046        353        692
```

Figura 31 – Resultado do comando para informações de memória

Nota-se que dos 1046 MB, apenas 353 MB são utilizados em condições de estabilidade. Para analisar o processamento e detalhes sobre o uso da memória utilizou-se o comando "ps aux". O resultado é apresentado na tabela 4.

Tabela 4 – Uso de memória e processamento do servidor

Resultados	Memória (MB)	Processamento (%)
Django	125.5	22.8
MySQL	144.3	0.2

Verifica-se que dos 353MB utilizados 269.8MB (76.4%) são utilizados pelo servidor (*Framework* + DBMS). Contudo a memória utilizada representa apenas 25% da memória total do servidor. Este resultado mostra o baixo uso de recursos do servidor, mesmo para um hardware limitado.

5.3.3 Limite de armazenamento de coordenadas

Nesse tópico deseja-se saber até quando o banco de dados consegue armazenar coordenadas dos motoristas, que em tese é responsável por maior parte do armazenamento.

Foi utilizado o comando "df -h" para obter informações gerais de armazenamento da RaspberryPi. A figura 32 mostra o resultado.

```
root@raspberrypi:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       6.7G  2.1G  4.3G  33% /
devtmpfs        214M     0  214M   0% /dev
tmpfs           218M     0  218M   0% /dev/shm
tmpfs           218M   13M  205M   6% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           218M     0  218M   0% /sys/fs/cgroup
/dev/mmcblk0p1   63M   21M   42M  33% /boot
```

Figura 32 – Resultado do comando para informações de armazenamento

Com base nas informações obtidas, existe um espaço ainda disponível de 5,2 GB. Esses dados foram obtidos num momento em que o banco de dados está com pouca informação, como poucos cadastros, pedidos e coordenadas. Dentre esses, as coordenadas possuem maior significância por existirem em quantidades muito maiores. Supondo que, desses 5.2 GB, sejam utilizados 4.0 GB para armazenamento de coordenadas.

Uma coordenada requisita 35 bytes para armazenamento. Logo é possível armazenar aproximadamente 114 milhões de coordenadas. Supondo que 100 motoristas, que trabalham 8horas/dia, enviam coordenadas a cada 10 segundos. Temos:

- Por minuto: 100 x 6 = 600 coordenadas
- Por hora: 600 x 60 = 36.000 coordenadas
- Por dia: 36000 x 8 = 288.800 coordenadas

Esse valor diário significa que o banco de dados é capaz de armazenar coordenadas por aproximadamente 395 dias.

6 Conclusão

Primeiramente, deve-se destacar que esse trabalho foi muito importante por trazer muito conhecimento da área de programação e tecnologia de informação. Muitos conhecimentos não vistos com muita profundidade na graduação foram necessários para a elaboração do projeto, como programação em Java, Python, sistemas embarcados e implementação de banco de dados.

Tratando-se dos aplicativos, optou-se por ser feito em Android por ser o mais utilizado pelos brasileiros, e foi feito através do Android Studio por ser a ferramenta mais versátil de desenvolvimento Android. Para o servidor, foi utilizado o *framework* Django por sua alta velocidade mesmo em *hardwares* mais simples e devido ao amplo suporte existente na Internet desse método. Optou-se por utilizar uma RaspberryPi para hospedá-lo por seu baixo custo e gasto de energia.

Os aplicativos e o servidor cumpriram todas as funções que foram propostas. Todas as etapas de gerenciamento, desde o cadastro da mercadoria até a sua entrega podem ser feitas através do conjunto estruturado formado por esse projeto. As principais funções, como otimização do trajeto do transportador (apesar de ser restrita apenas a 20 endereços) e o seu rastreamento puderam ser implementados com sucesso. Pode-se considerar as notificações aos clientes com previsão de horário de entrega como um diferencial por oferecer satisfação ao cliente ao prepará-los para o recebimento e funcionaram corretamente. Com exceção do cálculo da rota, todos os processos ocorrem muito rapidamente, em torno de milisegundos, o gasto de bateria do smartphone é muito baixo e o espaço para armazenamento necessário para os aplicativos é mínimo, não chegando na casa dos 10 MB.

Há margens para melhora, como a substituição do servidor embarcado para um mais robusto (que diminuiria o tempo gasto para o cálculo de rota e suportaria um maior número de usuários do aplicativo) e uma melhora no design, o qual não foi foco principal do trabalho.

6.1 Trabalhos Futuros

Com as informações obtidas é possível aplicar diversas funcionalidades. Por exemplo com a quantidade elevada de informações de GPS de cada veículo é possível fazer uma análise de Big Data para aprimorar a estimativa com base nos dados das entregas anteriores de cada motorista.

Outra funcionalidade que pode ser aplicada é atualizar a previsão do horário de entrega para o cliente quando o motorista se atrasar e quando for necessário colocar o

motivo. Pode ser implementado também no aplicativo do cliente um setor para entrar em contato com a transportadora para fazer reclamações, elogios ou sugestões.

Outro recurso que pode ser implementado é a superação das limitações do projeto de no máximo calcular o percurso para 20 endereços. Isto pode ser feito utilizando um algoritmo que faz mais de uma requisição no Google Maps API e com base nos resultados calcular o melhor trajeto.

Além disso o aplicativo do motorista e do cliente foram apenas desenvolvidos para Android. Em trabalhos futuros podem ser desenvolvidos para iOS também.

Referências

27ª Pesquisa Anual do Uso de TI, 2016. Meirelles, F. S., 2016. Disponível em: <<http://eaesp.fgvsp.br/sites/eaesp.fgvsp.br/files/pesti2016gvciappt.pdf>>. Acesso em: 11 nov. 2017. 21

4 Conceitos sobre REST que Qualquer Desenvolvedor Precisa Conhecer. [s.n.], 2016. Disponível em: <<http://blog.algaworks.com/4-conceitos-sobre-rest-que-qualquer-desenvolvedor-precisa-conhecer/>>. Acesso em: 11 nov. 2017. 26

95,5% dos smartphones vendidos no Brasil são Androids. Higa, P., 2016. Disponível em: <https://tecnoblog.net/203749/android-ios-market-share-brasil-3t-2016/>. Acesso em: 11 nov. 2017. 36

DJANGO Overview. [s.n.], 2017. Disponível em: <https://djangobook.com/tutorials/django-overview/>. Acesso em: 11 nov. 2017. 15, 33

HELLMAN, R. A. F. **Aplicativo android e website interativo para busca de menores preços de produtos com código de barras**. São Carlos: EESC, 2016. 27

INFORMATION capacity and versions of the QR Code. [s.n.], 2017. Disponível em: <http://www.qrcode.com/en/about/version.html>. Acesso em: 11 nov. 2017. 27

MICRODADOS TIC Domicílios - 2015 - Indivíduos. Cetic.br, 2016. Acesso e uso das TIC nos domicílios. Disponível em: <http://cetic.br/arquivos/domicilios/2016/individuos/>. Acesso em: 11 nov. 2017. 21

PERES, N. **Relatório Final de Iniciação Científica: GPS, posicionamento e relógios precisos**. São Carlos: [s.n.], 2016. 29

PERES N.; OKUYAMA, G. H. L. P. W. **Projeto ZEIT**. São Carlos: [s.n.], 2016. 31 p. 21

SAASY maps. Chickermane, A., 2016. Disponível em: <https://www.slideshare.net/AnushaChickermane/saasy-maps>. Acesso em: 11 nov. 2017. 15, 33

SOFTWARE Java. [s.n.], 2017. Disponível em: <https://www.oracle.com/br/java/index.html>. Acesso em: 11 nov. 2017. 38

WELCOME to Raspbian. [s.n.], 2017. Disponível em: <http://www.raspbian.org>. Acesso em: 11 nov. 2017. 32