CENTRO UNIVERSITÁRIO
Fundação Santo André

Arthur Cicuto Pires

Victor Vieira Paulino

Sistema de acompanhamento de transporte público para deficientes visuais

Santo André

2017

Arthur Cicuto Pires
Victor Vieira Paulino

Sistema de acompanhamento de transporte público para deficientes visuais

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia Engenheiro Celso Daniel do Centro Universitário Fundação Santo André, como exigência parcial para obtenção do grau de Bacharel em Engenharia de Computação.

Centro Universitário Fundação Santo André – CUFSA

Engenharia de Computação com Ênfase em Software

Orientador: Prof. Dr. Marcos Forte

Santo André

2017

ARTHUR CICUTO PIRES
VICTOR VIEIRA PAULINO

Sistema de acompanhamento de transporte público para deficientes visuais

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia Engenheiro Celso Daniel do Centro Universitário Fundação Santo André, como exigência parcial para obtenção do grau de Bacharel em Engenharia de Computação.

Trabalho aprovado. Santo André, 2017:

Prof. Dr. Marcos Forte
CUFSA

Nome 1
CUFSA

Nome 2
CUFSA

Santo André
2017

Agradecimentos

Agradecimentos aqui

Resumo

Colocar o resumo aqui.

Palavras-chave: Acessibilidade, deficientes visuais, transporte público, aplicativo mobile

Abstract

Abstract here. **Keywords**: Acessibility, smartphone, bus, application.

Sumário

Lista de ilustrações

Lista de abreviaturas e siglas

CUFSA Centro Universitário Fundação Santo André

1 Introdução

1.1 Referências do Sistema

Smartphones tem se tornado cada vez mais presentes na vida das pessoas. Uma pesquisa realizada pelo FGV-SP em 2016 [MEIRELLES, 2016] demonstrou que o número de aparelhos chegou a 168 milhões só no Brasil. Com sua facilidade de acesso, surgem inúmeras soluções que resolvem problemas do dia-a-dia dos usuários.

Dentre essas soluções, aplicações para smartphones que ajudam na mobilidade são cada vez mais comuns. Os aplicativos CittaMobi [VIEIRA, 2015] e Moovit [GOMES, 2015] vieram para mostrar que a tecnologia embarcada nos aparelhos podem ajudar a prever quanto tempo falta para o ônibus chegar em um ponto de parada, em tempo real. Eles capturam a geolocalização do usuário para saber qual ponto de ônibus eles estão próximos, possibilitando o usuário dizer de forma mais rápida qual seu ponto. Informando ao aplicativo qual seu ponto, eles podem selecionar um ônibus que passa no ponto selecionado, para saber quanto tempo resta para o veículo chegar.

Isso ajuda os usuários a se programar melhor, possibilitando a pessoa sair em um horário mais oportuno ou deixando ela mais tranquila sabendo que em breve seu ônibus chegará.

1.2 Descrição Geral

Este trabalho visa facilitar a vida de deficientes visuais que utilizam ônibus como meio de transporte. O aplicativo proposto irá possibilitar ao deficiente visual saber quanto tempo falta para seu ônibus chegar, enquanto o sistema se encarrega de avisar o motorista do ônibus qual o próximo ponto onde terá um deficiente visual esperando por aquele ônibus.

Sistemas operacionais de smartphone, como Android e iOS, possuem ferramentas nativas que adaptam o uso de aplicativos para pessoas com deficiências, possibilitando a utilização do aparelho sem grandes dificuldades, mas, nem sempre, criam boas experiências de uso.

O Android possui a ferramenta Talkback, para auxiliar no uso de qualquer aplicativo. Ao desenvolver uma solução para o sistema, é possível colocar tags específicas em cada elemento da

tela da sua aplicação. Isso possibilita o Talkback ler a tela com maiores detalhes para o deficiente visual ou utilizar a função de áudio dele para fazer áudios descrições mais detalhadas sobre o significado de uma tela.

Fazer aplicativos que funcionem em conjunto com essas tecnologias voltadas a deficientes já disponíveis, não é um trabalho difícil, mas criar boas experiências de uso que facilitem a vida de deficientes visuais é uma grande tarefa a ser cumprida.

Por isso é necessário adicionar outras tecnologias que facilitem o uso do app, neste caso, os Beacons. Beacon é um dispositivo que utiliza Bluetooth 4.0 (que tem baixo consumo de energia). Se existe um smartphone próximo a um Beacon, o aplicativo pode informar sua localização com maior precisão que um GPS.

Dessa forma quando um deficiente visual chegar no ponto, ele abre o aplicativo e, com uso do Beacon instalado no ponto, nosso aplicativo sabe em qual ponto o cego está. Sabendo isso, o app lista quais ônibus passam ali. Após o deficiente visual escolher um dos ônibus, o aplicativo vai notificar em intervalos pré-definidos quanto tempo falta para o ônibus chegar, em contrapartida o sistema irá alertar o motorista quando ele estiver próximo ao ponto em que existe um deficiente visual esperando por ele.

1.3 Restrições de projeto

- O smartphone deve ter o sistema operacional Android 4.1 (API Level 16) ou posterior instalado;
- O smartphone deve possuir Bluetooth 4.0 LE ou superior;
- O smartphone deve estar com a função Talkback ativada;
- O ônibus deve prover sinal de rede Wi-Fi para que o módulo do ônibus possa se comunicar.

2 Tecnologias Utilizadas

2.1 Comunicação

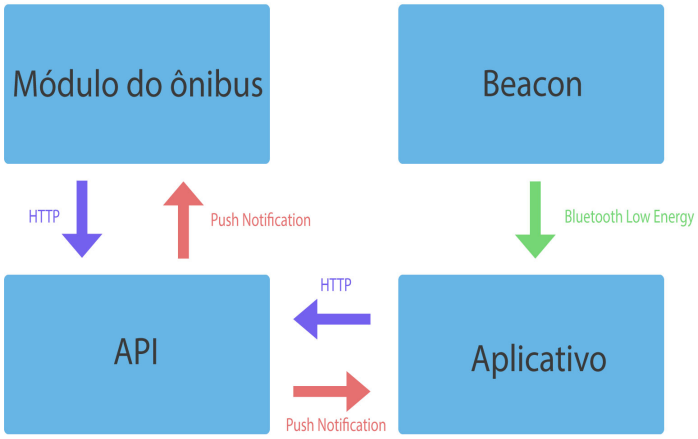

Figura 1 – Protocolos de comunicação utilizados.

2.1.1 HTTP

O Protocolo de transferência de hipertexto (do inglês *Hypertext Transfer Protocol*), é um protocolo para trafego de informações pela internet entre cliente e servidor. Sempre que o cliente deseja enviar algo para o servidor, é enviado uma requisição *HTTP* contendo nela um cabeçalho, com informações sobre a requisição, e o corpo, contendo uma mensagem a ser entregue. Uma requisição *HTTP* pode ser do tipo GET, POST, PUT, DELETE, HEAD, TRACE, OPTIONS, ou CONNECT, cada uma para que o servidor saiba a intenção do cliente e saiba lidar com o corpo da mensagem. No cabeçalho do protocolo contém o campo *Content-Type* com o tipo de *MIME* contido no corpo da requisição.

O tipo MIME (*Multipurpose Internet Mail Extensions* - Extensões de correio de Internet multifunções) é um padrão proposto pelos laboratórios Bell Communications, em 1991, para aumentar a possibilidade de inserir documentos (imagens, sons, texto, etc.) em uma mensagem. Desde então, o tipo MIME é usado para formatar tanto documentos anexos em uma mensagem

quanto os transferidos pelo protocolo HTTP. (CCM br.ccm.net, 2017, Formatos e extensões de arquivos - Tipo MIME).

2.1.2 Push Notification

Push Notification é um serviço de entrega de mensagens, parecido com *SMS* (*Shot Message Service*), que utiliza a internet para isso. Um caso comum do uso de *push notifications* é ter a necessidade de notificar um ou mais usuários de um aplicativo sobre algo, não sendo necessário que a aplicação esteja aberta. Alguns serviços são especializados na entrega de mensagens desse tipo, como o *Firebase Push Notification*, que provê um *SDK* para que o desenvolvedor implemente o serviço de recebimento destas mensagens na sua aplicação para quando for preciso enviar uma mensagem, faça apenas requisições *HTTP* para o serviço e ele se encarregue de entregar a mensagem para os dispositivos disponíveis.

2.1.3 Beacons

O Beacon é um pequeno dispositivo que utiliza uma tecnologia chamada Bluetooth Low Energy (BLE), que emite um sinal intermitente de ondas de rádio que consegue localizar seu smartphone em um determinado raio.(CARNEIRO, Conrado. Use Mobile. Maio 27, 2016. <http://usemobile.com.br/conheca-beacon/>. A grande novidade dos aparelhos beacons, além do custo acessível, é que eles podem ser instalados em paredes, produtos ou vitrines, permitindo a comunicação entre empresa e público por meio da localização e sem a necessidade de acesso à internet, já que utiliza o bluetooth do smartphone para enviar as mensagens. (SAES, Bruno. Impacta. Dez 09, 2014. <http://www.impacta.com.br/blog/2014/12/09/o-que-sao-beacons-como-mudarao-rotina/> (10.10.17)

2.1.4 Realtime Database

O Firebase Realtime Database é um banco de dados hospedado na nuvem. Os dados são armazenados como JSON e sincronizados em tempo real com todos os clientes conectados. Em vez de solicitações HTTP típicas, o Firebase Realtime Database usa a sincronização de dados. Sempre que os dados são alterados, todos os dispositivos conectados recebem essa atualização em milissegundos. (Firebase. <https://firebase.google.com/docs/database/?hl=pt-br>) (10.10.17)

2.2 Hardware

2.2.1 Raspberry Pi 3

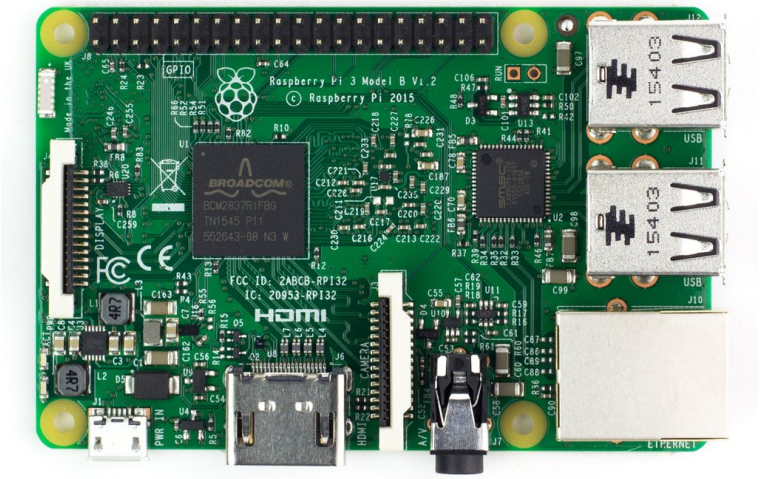

Figura 2 – Raspberry 3.

"A Raspberry Pi é uma máquina completa, com considerável poder de processamento, em uma placa de circuito impresso menor do que um cartão de crédito. Com ela você pode ter resultados impressionantes."(Upton, E. and Halfacree, G., 2017, Raspberry Pi - Manual do Usuário). Esta pequena placa permite ter um computador em um pequeno espaço, contando com conectividades como *Bluetooth* e *WiFi*, é uma excelente opção para projetos que necessitam de mais poder de processamento em um pequeno espaço.

A Fundação Raspberry Pi, responsável pelo seu desenvolvimento, também mantém o foco da placa no meio educacional, apresentando uma plataforma acessível para que cada vez mais pessoas se interessem por desenvolvimento de softwares.

Especificações:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM

- BCM43438 wireless LAN e Bluetooth Low Energy (BLE)
- 40-pin extended GPIO
- 4 USB 2 ports
- 4 Pole stereo output e composite video port
- Full size HDMI
- CSI camera port para conectar uma câmera Raspberry Pi
- DSI display port para conectar um display touchscreen Raspberry Pi
- Micro SD port para carregar o sistema operacional e armazenar dados

2.2.2 HM-10

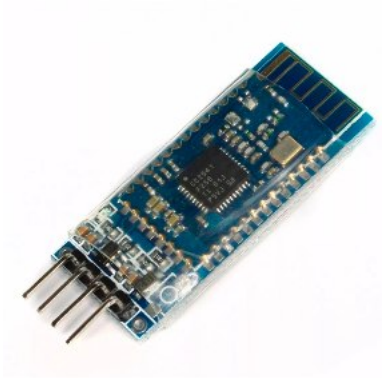

Figura 3 – Módulo Bluetooth HM-10.

A empresa Jinan Huamao é responsável pelo desenvolvimento do HM-10, um pequeno módulo *Bluetooth* 4.0 BLE baseado no chip CC2540 que trabalha com alimentação de 3.3V. Com ele é possível realizar comunicações utilizando o protocolo *Bluetooth*.

2.2.3 Arduino Uno

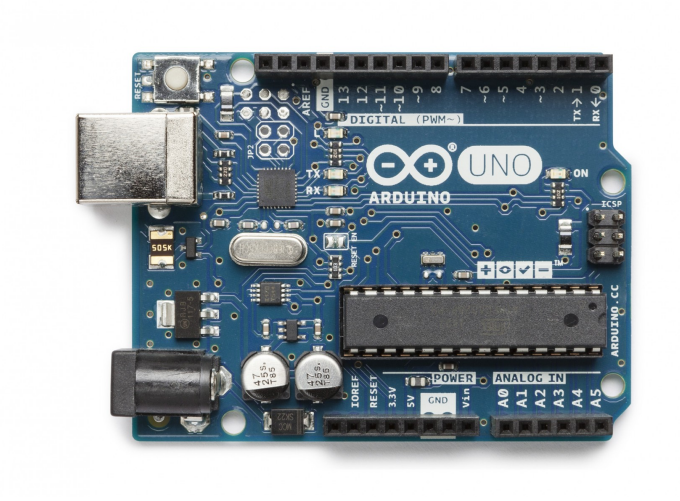

Figura 4 – Módulo Bluetooth HM10.

Arduino é uma plataforma de prototipagem desenvolvida em 2005, na Itália. Conectando a placa via USB em um computador, é possível programar o microcontrolador Atmel que ela possui, tendo a possibilidade de controlar dispositivos externos por meio de suas portas *GPIO (General Purpose Input/Output)*.

O Arduino pode ser utilizado para desenvolver objetos interativos independentes, ou pode ser conectado a um computador, a uma rede, ou até mesmo à Internet para recuperar e enviar dados do Arduino e atuar sobre eles. (MCROBERTS, Michael. Arduíno Básico. Novatec, 2011. Cap. 1, p. 23)

2.2.4 NEO 6M

Figura 5 – Módulo NEO u-blox 6 GPS.

O NEO U-BLOX 6 é um módulo que permite obter a geolocalização por meio de *GPS (Global Position System)*. A tecnologia GPS funciona através de ondas de rádio que são transmitidas por satélites, onde o aparelho receptor localiza o sinal de pelo menos quatro deles. Com o sinal recebido, é feito cálculos para obter a localização no globo.

Realizando uma comunicação com protocolo UART (Universal Asynchronous Receiver/-Transmitter), é possível obter as informações do módulo. Tais informações obtidas seguem a especificação NMEA, que define como os dados serão transmiitidos

FALAR MAIS SOBRE UART E NMEA

2.3 Software

2.3.1 Sistemas Operacionais

2.3.1.1 Android

Existem vários sistemas operacionais voltados para dispositivos móveis, sendo Android o mais popular. O sistema foi desenvolvido pela empresa Android Inc, posteriormente adquirida pela Google Inc. Foi lançado em 2008 e conta com o apoio da Open Handset Alliance (um conjunto de empresas que atuam juntas para o melhoramento dos padrões da telefonia e também do desenvolvimento do sistema Android). Tal sistema é muito acessível e popular. Está disponível para uma enorme variedade de dispositivos. O Android tem como base o sistema operacional Linux, conhecido por ser um sistema flexível, adaptável a várias arquiteturas de

processador, seguro e eficiente. Um dos grandes diferenciais do Android e que contribui para sua popularidade, é o fato de que o sistema é compatível com vários hardwares e pode estar disponível em smartphones de diversos fabricantes. (Viva O Android <https://www.vivaoandroid.com.br/android/> (11.10.17)

Segundo uma pesquisa feita pela IDC (*International Data Corporation*) no primeiro quadrimestre de 2017, o Android se mostra um sistema consolidado no mercado, tendo participação de 85% do mercado. O gráfico a seguir mostra que desde 2014 é o sistema operacional mais utilizado em dispositivos móveis.

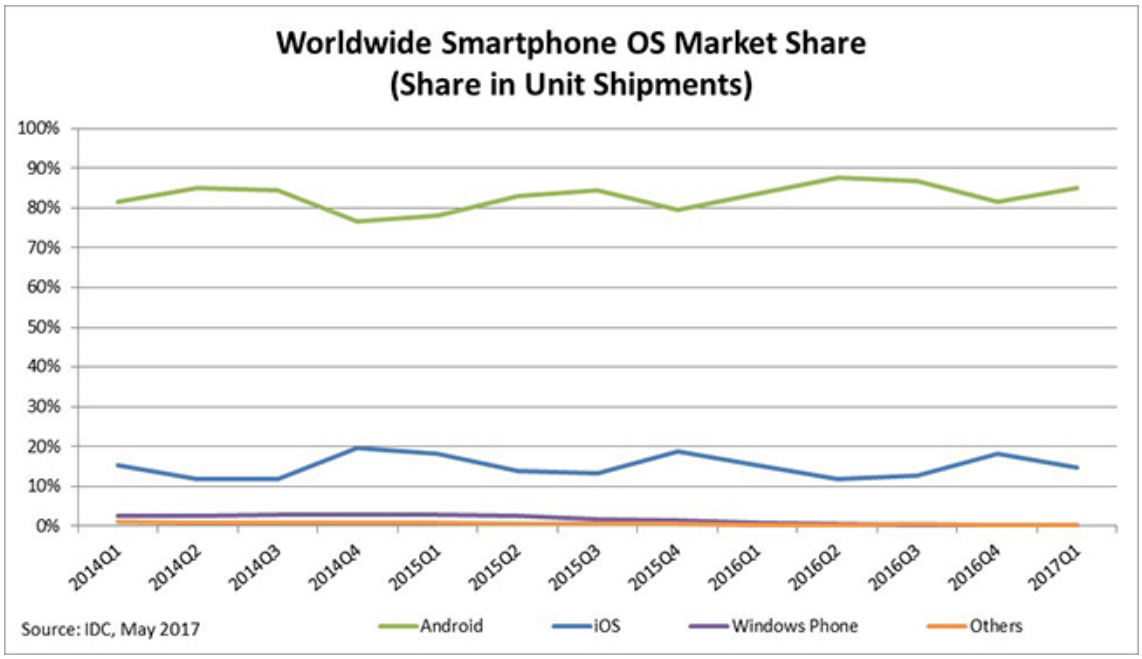

Figura 6 – Pesquisa realizada pela IDC.

2.3.1.2 Android Things

A comprehensive way to build IoT products with the power of Android, one of the world's most supported operating systems. Now any Android developer can quickly build a smart device using Android APIs and Google services, while staying highly secure with updates direct from Google. (PIEKARSKI, Wayne. Android Developers. Dez 13, 2016. <https://android-developers.googleblog.com/2016/12/announcing-googles-new-internet-of-things-platform-with-weave-and-android-things.html>) O Android Things foi criado voltado ao mercado *IoT (Internet of Things)*. Foi criado a partir da versão do Android para *smartphones*, o que trouxe quase todas APIs disponíveis. Utilizando a mesma *IDE* para desenvolvimento, qualquer desenvolvedor

Android pode desenvolver soluções *IoT* com os conhecimentos que já possui.

Atualmente o sistema tem suporte para rodar nas seguintes plataformas:

- NXP Pico i.MX7D
- NXP Pico i.MX6UL
- NXP Argon i.MX6UL
- NXP SprIoT i.MX6UL
- Raspberry Pi 3

2.3.1.3 Linguagem de Programação

A Google adicionou o Kotlin à limitada lista de linguagens de programação suportada para o desenvolvimento Android. Até então, apenas Java e C++ faziam parte dessa lista. Dentre as razões para a escolha do Kotlin, a Google mencionou o fato da linguagem "ser concisa, expressiva e projetada para ser type-safe e null safe"e também o fato de que "vários desenvolvedores Android também consideram que Kotlin torna o desenvolvimento mais ágil e divertido". Outra razão importante é o fato do Kotlin ser uma linguagem que se integra totalmente ao Java e roda na JVM. Além disso, também é possível invocar código em C++/Android já que ela suporta JNI por meio de modificadores de acesso externo no código-fonte. A partir do código-fonte do Kotlin é possível gerar bytecode para a JVM ou código-fonte Javascript. (AVRAM, Abel. InfoQ. Jun 03, 2017. <https://www.infoq.com/br/news/2017/06/android-kotlin>)

2.3.2 Backend

2.3.2.1 Linguagem

Para o desenvolvimento do *web service* foi escolhido um conjunto de tecnologias *server-side* baseadas em JavaScript.

2.3.2.1.1 JavaScript

JavaScript é uma linguagem de programação *client-side*, utilizada para manipular os comportamentos de uma página, controlando o HTML e o CSS. Outra característica dela, é o fato

de ser uma linguagem orientada à eventos. Para explicar melhor o que são eventos, é importante citar que uma página HTML utiliza tags para representar seus elementos, podendo conter menus, botões e formulários, entre outros, em seu corpo. Cada elemento possui alguns atributos, sendo alguns desses chamados atributos de eventos, como por exemplo o *onClick* que realiza alguma função caso o elemento referente seja clicado pelo usuário. Para dizer as ações que devem ser tomadas quando um evento é acionado, pode-se utilizar o JavaScript. Com o decorrer do tempo, foram desenvolvidas algumas modificações em cima de JavaScript para possibilitar a utilização do mesmo no *server-side*, possibilitando o desenvolvimento de um *web service* em torno de uma mesma linguagem.

2.3.2.2 MongoDB

É um banco de dados não relacional com uma escalabilidade muito boa. Ele utiliza conceitos de *collections* e *documents* em sua construção. As *collections* são equivalentes aos bancos de um ambiente que utiliza o SQL. Já os *documents*, se equivalem aos registros de cada banco. Os dados são guardados em arquivos similares aos de formato JSON (*JavaScript Object Notation*). Outro item importante sobre o MongoDB é o fato de ser *schemaless*, tornando-o bem flexível em relação a inclusão de dados diferentes em uma mesma *collection*, fazendo com que a validação de dados fique nas mãos dos desenvolvedores. Apesar de *schemaless*, é possível criar *schemas* para auxiliar no desenvolvimento. Ao utilizar o mongoose, ferramenta desenvolvida em cima do MongoDB para trabalhar nele como se estivesse utilizando um banco relacional, é possível definir de antemão, quais os atributos que devem existir em cada *collection* necessária para a aplicação.

2.3.2.3 Express.js

É um framework para Node.js que ajuda na organização de sua aplicação, caso use a arquitetura MVC, no lado do servidor. Uma de suas funções é a de facilitar a criação e manutenção de rotas, realizando uma configuração inicial com os caminhos para os *controllers*, *models* e *views* utilizados pela sua aplicação, além de informar os dados de inicialização do servidor. Além disso, ele gerencia todos os *middlewares*, permitindo o fácil acesso a isso tudo em qualquer arquivo relacionado a aplicação.

2.3.2.4 Node.js

Plataforma principal para o funcionamento da aplicação, construída sobre o motor Javascript V8 do Google Chrome. O Node.js foi desenvolvido para construir aplicações web escaláveis de forma assíncrona, cuidando de várias conexões de maneira concorrente. Ele consegue isso através da utilização de callbacks. Sempre que ocorre um evento, é disparado um callback que dirá o que deve ser realizado pela aplicação. Esse processo é feito para todas as funções presentes na aplicação e sempre que houver um evento sendo disparado. Como no lado do servidor, não existe uma interface gráfica a ser visualizada, os eventos nesse caso são dados de I/O, como algum parâmetro usado em uma busca no banco de dados, a resposta para aquele parâmetro, entre outros casos. O Node.js é uma plataforma extremamente modularizada, com módulos criados por desenvolvedores ativos no mundo todo. Para gerenciar esses módulos, ele utiliza o npm, responsável por instalar, atualizar ou remover suas dependências. É o Node.js que realiza a conexão com servidores e diz quais bancos de dados serão utilizados pela aplicação, na configuração inicial.

2.3.2.5 REST

Segundo [ALMEIDA, 2016] O padrão REST utiliza um conjunto de operações aplicáveis a todos os recursos de informação utilizando o protocolo HTTP. Suas operações mais utilizadas são DELETE, POST, PUT e GET. No padrão REST, cada recurso possui um identificador único, representado, por exemplo no HTTP, pela sua URL de acesso. Dentre os princípios do padrão REST, segundo [GOMES, 2009] é importante dar um foco em cinco. São eles:

Dar um identificador a todas as coisas Considerando as coisas como sendo os recursos, o principal benefício desse item é a utilização de URIs que identifiquem o necessário, especificando se os recursos que sua aplicação oferece são itens individuais, conjuntos de itens, objetos virtuais e físicos, ou resultados de computação.

Vincular as coisas Esse princípio faclita a utilização de serviços externos em sua aplicação. A vinculação, nesse caso, é feita através de links para que o servidor saiba aonde estão alocados os recursos necessários. Esse prinncípio é extremamente útil para o desenvolvimento de aplicações dinâmicas.

Utilizar métodos padronizados Aqui entram os verbos HTTP, como GET, PUT, POST e DELETE. Esses verbos representam métodos padrões para que a aplicação web saiba o que

deve ser feito com determinada URI. Essa padronização faz com que seu navegador saiba exatamente o que fazer.

Recursos com múltiplas representações Este princípio diz para oferecer formas diversas dos recursos para diferentes finalidades.

Comunicar sem estado Manter o estado da comunicação no cliente ou transformá-lo no estado do recurso. Isso faz com que a escalabilidade da aplicação seja um dos focos desse princípio. Outro aspecto desse tópico, é o isolamento do cliente em relação ao servidor, fazendo com que o cliente não fique preso ao mesmo servidor para realizar duas solicitações consecutivas, com isso, ele recebe as informações do servidor sem precisar se preocupar se está ocorrendo alguma troca de disco rígido ou atualização de software no servidor.

2.3.3 Referências

Atributos de eventos

Guia introdutório sobre JavaScript

ALMEIDA, Flávio. MEAN - Full stack JavaScript para aplicações web com MongoDB, Express, Angular e Node. ed. Casa do Código, 2016.

O que é Node.js?

GOMES, André Faria apud TIKOV, Stefan. InfoQueue. "Uma rápida introdução ao REST". 29 de outubro de 2009. <https://www.infoq.com/br/articles/rest-introduction>(1.10.17)

3 Desenvolvimento

3.1 Descrição da Informação

3.1.1 Visão Geral

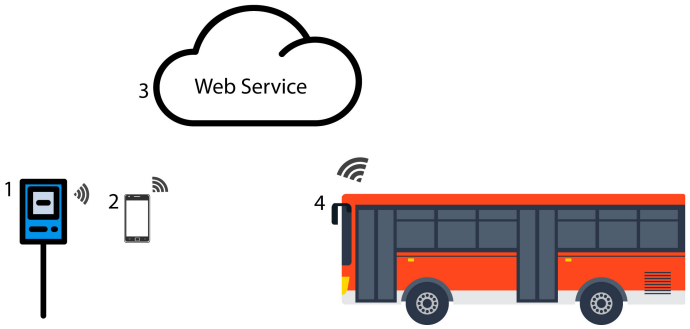

Figura 7 – Visão geral da comunicação dos componentes.

1. Módulo da parada de ônibus: Emite informações de identificação da parada.

2. Aplicativo: Reconhece o ponto de ônibus e solicita informações do *Web Service*.

3. Web Service: Intermediário entre o aplicativo e o módulo do ônibus.

4. Módulo do Ônibus: Mantém constante comunicação com o *Web Service* enviando dados de geolocalização. Recebe também informação se deve alertar o motorista sobre deficiente visual na próxima parada.

3.1.2 Representação do Fluxo da Informação

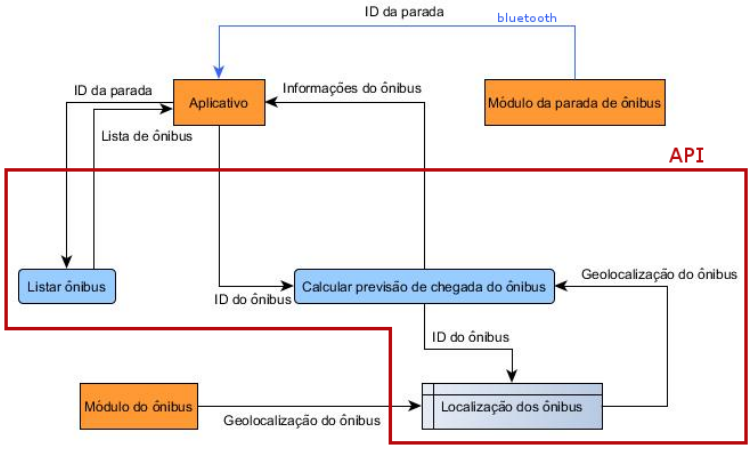

Figura 8 – Diagrama de fluxo de dados.

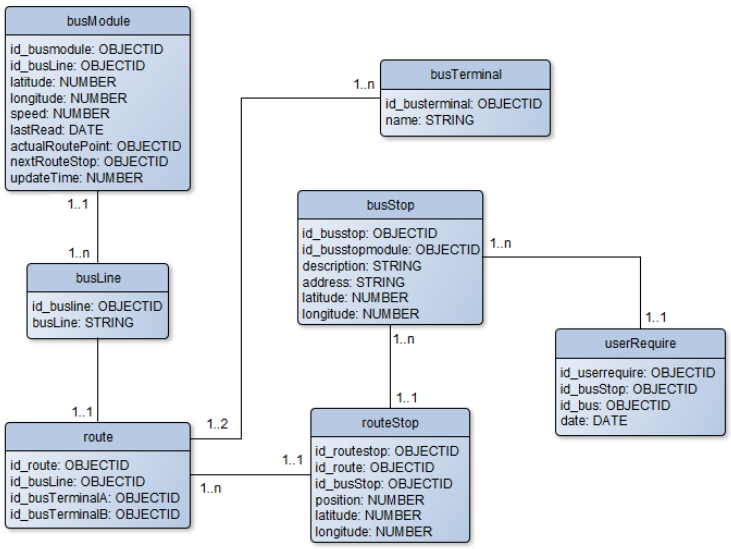

Figura 9 – Diagrama de banco de dados.

3.1.3 Descrição Funcional

3.1.3.1 Aplicativo para dispositivo móvel

Aplicativo que irá interagir com o deficiente visual. Sua função é verificar qual Beacon está mais próximo para que a API possa saber sua localização, podendo listar, via interface áudio-visual, para o usuário, quais linhas passam no ponto de parada que ele está.

3.1.3.2 API

Sistema que recebe informações do aplicativo e do módulo do ônibus. Tem como objetivo acessar os dados gravados no banco de dados para que possa prover informações de previsão ao aplicativo. Também é responsável por verificar se o módulo do ônibus deve alertar a presença de um usuário no próximo ponto. Além de calcular a previsão de um ônibus até o ponto de parada selecionado.

3.1.3.3 Módulo do ponto de ônibus

Dispositivo localizado em um determinado ponto de parada de ônibus. Emite constantemente um sinal ID para a identificação do ponto que ele se refere.

3.1.3.4 Módulo do ônibus

Dispositivo instalado no ônibus. Mantém comunicação constante com a API para informar sua geolocalização. Verifica ao mesmo tempo a necessidade de alertar o motorista se existe um deficiente visual aguardando no próximo ponto de parada.

3.1.4 Interfaces com Sistema

3.1.4.1 Aplicativo

3.1.4.1.1 Busca por um ponto próximo

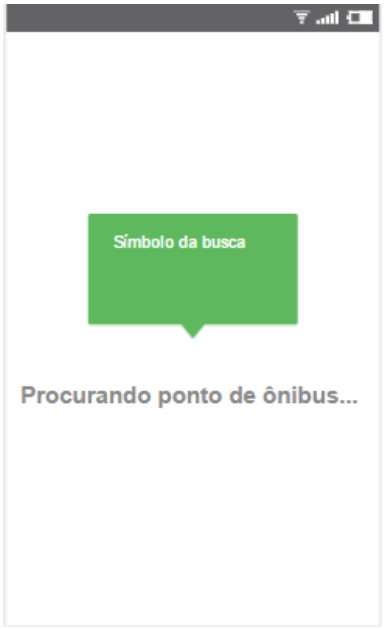

Figura 10 – Tela do aplicativo ao buscar por um ponto de ônibus próximo.

Número	Nome	Descrição	Requisitos	Grupo
1	Símbolo da busca do ponto de ônibus	Indica que o aplicativo está procurando um ponto de ônibus	-	Imagem e texto
2	Áudio sobre busca	Indica ao usuário que está sendo feito uma busca por algum ponto próximo	Função Talkback ativa	Áudio

Tabela 1 – Descrição dos elementos da tela de busca por ponto de ônibus próximo.

3.1.4.1.2 Busca por um ponto na API

Figura 11 – Tela do aplicativo ao buscar por um ponto de ônibus na API.

Número	Nome	Descrição	Requisitos	Grupo
1	Símbolo da busca das linhas de ônibus	Indica que o aplicativo procura as linhas de ônibus	O sistema deve ter detectado um ponto de ônibus	Imagem e texto
2	Áudio sobre busca	Indica ao usuário que está sendo feito uma busca dos ônibus disponíveis	Função Talkback ativa	Áudio

Tabela 2 – Descrição dos elementos da tela de busca por ponto de ônibus na API.

3.1.4.1.3 Lista de ônibus disponíveis

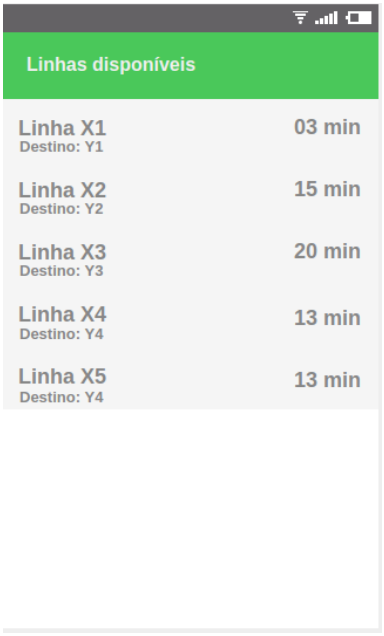

Figura 12 – Tela do aplicativo com ônibus disponíveis.

Número	Nome	Descrição	Requisitos	Grupo
1	Lista de linhas	Lista de linhas que o usuário pode escolher	Ter recebido uma lista da API	Botão
2	Áudio sobre escolha de um item	Indica que a lista de ônibus já está disponível	Função Talkback ativa	Áudio

Tabela 3 – Descrição dos elementos da tela de busca por ponto de ônibus na API.

3.1.4.1.4 Detalhes do ônibus

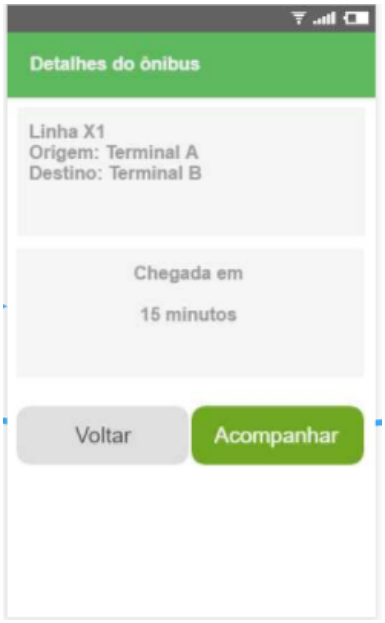

Figura 13 – Tela do aplicativo com detalhes de um ônibus.

Número	Nome	Descrição	Requisitos	Grupo
1	Linha X1	Mostra a linha selecionada	Receber previsão da API	Texto
2	Origem	Exibe o ponto inicial da linha	Receber previsão da API	Texto
3	Destino	Exibe o ponto final da linha	Receber previsão da API	Texto
4	Chegada em	Exibe a previsão de chegada da linha	Receber previsão da API	Texto
5	Voltar	Volta para a seleção de linhas	Receber previsão da API	Botão
6	Solicitar ônibus	Solicita que o ônibus pare no seu ponto e acionar o acompanhamento dele	Receber previsão da API	Botão
7	Áudio sobre previsão	Alerta ao usuário a previsão do ônibus	Receber previsão da API	Áudio

Tabela 4 – Descrição dos elementos da tela de detalhes do ônibus.

3.1.4.1.5 Aguardando um ônibus

Figura 14 – Tela do aplicativo sobre a solicitação de um ônibus.

Número	Nome	Descrição	Requisitos	Grupo
1	Informação de previsão	O sistema irá informar o usuário até a chegada do ônibus	Ter selecionado botão Solicitar ônibus	Texto
2	Favoritar	Adiciona ônibus como favorito	O ônibus não pode estar cadastrado como favorito. Caso esteja o botão não é exibido	Botão
3	Cancelar	Cancela o acompanhamento do ônibus e solicita que não pare mais no ponto	Ter selecionado botão Solicitar ônibus	Botão
4	Áudio sobre o acompanhamento	Informa ao usuário que está sendo feito o acompanhamento do ônibus	Ter escolhido acompanhar um ônibus. Função Talkback ativa	Áudio

Tabela 5 – Descrição dos elementos da tela sobre a solicitação de um ônibus.

3.1.4.2 Módulo do ônibus

3.1.4.2.1 Carregando Informações

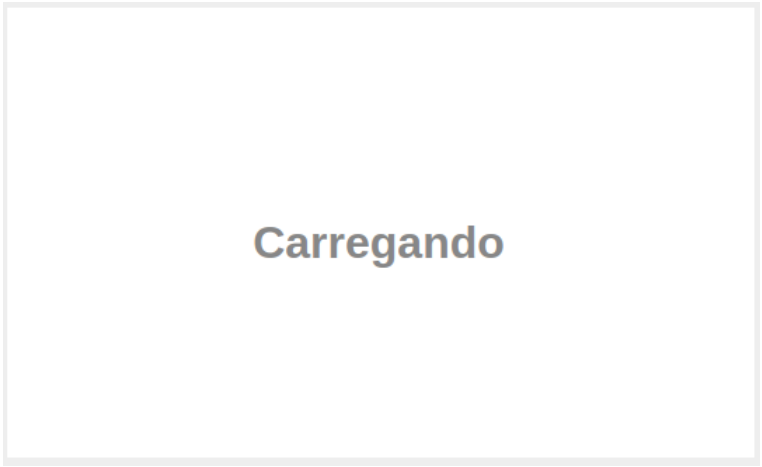

Figura 15 – Tela do módulo do ônibus quando está carregando informações.

Número	Nome	Descrição	Requisitos	Grupo
1	Mensagem	Informa que está carregando informações do servidor	Ter iniciado o sistema	Texto

Tabela 6 – Descrição dos elementos da tela sobre estar carregando informações.

3.1.4.2.2 Aguardando iniciar a rota

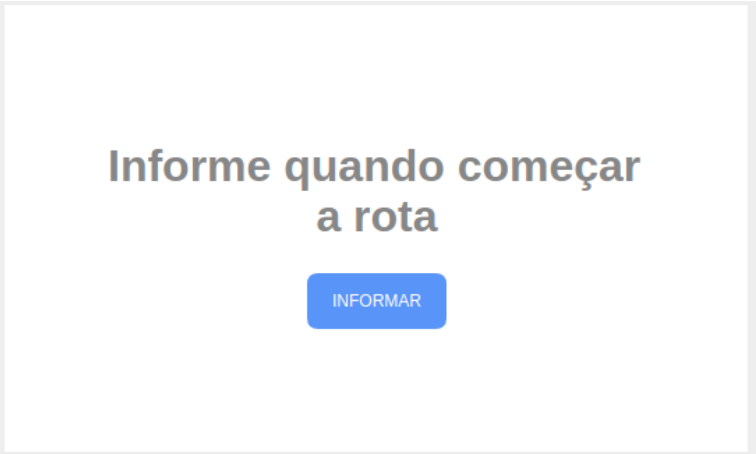

Figura 16 – Tela do módulo do ônibus aguardando motorista inicar rota.

Número	Nome	Descrição	Requisitos	Grupo
1	Informação para começar rota	Informar que precisa avisar quando começar a rota	-	Texto
2	Começar rota	Começar rotina de envio da posição	-	Botão

Tabela 7 – Descrição dos elementos da tela sobre começar uma rota.

3.1.4.2.3 Realizando uma rota

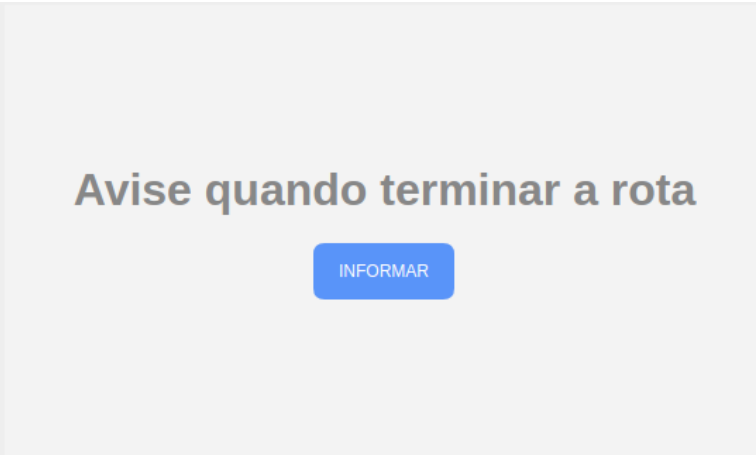

Figura 17 – Tela do módulo do ônibus quando motorista está fazendo a rota.

Número	Nome	Descrição	Requisitos	Grupo
1	Informação sobre termino da rota	Informar que precisa avisar quando terminar a rota	Ter iniciado uma rota	Texto
2	Terminar rota	Terminar rotina de envio da posição	Ter iniciado uma rota	Botão

Tabela 8 – Descrição dos elementos da tela sobre terminar uma rota.

3.1.4.2.4 Aviso para parar no próximo ponto

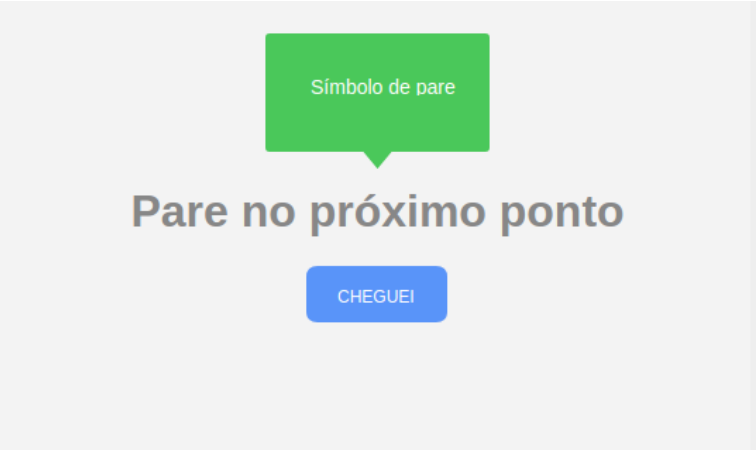

Figura 18 – Tela do módulo do ônibus quando motorista precisar parar no próximo ponto.

Número	Nome	Descrição	Requisitos	Grupo
1	Simbolo de alerta para parar	Informar que precisa parar no próximo ponto	Servidor informar a necessidade	Imagem e Texto
2	Informar chegada	Envia informação que chegou no ponto	Ter chegado no ponto	Botão

Tabela 9 – Descrição dos elementos da tela sobre terminar uma rota.

3.1.5 Casos de Uso

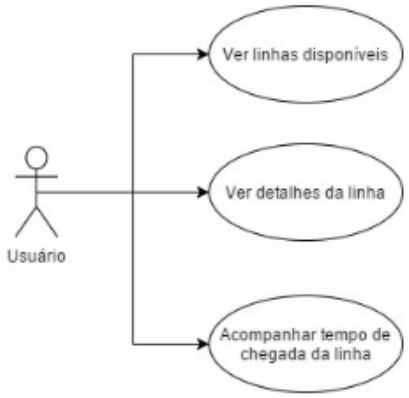

Figura 19 – Diagrama de caso de uso.

3.1.5.1 Narrativas: Casos de Uso

Solicitar horário do próximo ônibus da linha e sentido escolhido: Este caso de uso acontece quando um usuário solicita qual será a previsão de horário do próximo ônibus, de uma linha e sentido que ele poderá escolher de acordo com o seu ponto de ônibus.

Solicitar parada do ônibus escolhido: Este caso de uso é uma extensão do caso de uso Solicitar horário do próximo ônibus da linha e sentido escolhido, onde depois de escolher uma linha e sentido ele poderá solicitar a parada do próximo ônibus escolhido.

3.1.5.2 Diagramas de apoio para compreensão funcional

Identificação: UC001	
Nome: Solicitar horário do próximo ônibus	
Atores: Usuário	
Pré-condições: O aplicativo precisa ter lido o ID do módulo do ponto de ônibus	
Pós-condições: Retorno do horário do próximo ônibus e da solicitação de parada	
Fluxo de eventos	
Ator	**Sistema**
1. Usuário chega ao ponto de ônibus	2. Sistema lê o ID do ponto de ônibus e retorna uma lista de linhas
3. Usuário escolhe uma linha	4. Informar constantemente o horário do ônibus
Fluxo alternativo	
Não possui fluxo alternativo	

Tabela 10 – Tabela com caso de uso UC001.

Identificação: UC002	
Nome: Solicitar parada do ônibus escolhido	
Atores: Usuário	
Pré-condições: O usuário precisa ter solicitado o ônibus de uma linha	
Pós-condições: Confirmação de parada	
Fluxo de eventos	
Ator	**Sistema**
1. Usuário confirma solicitação de parada no seu ponto	2. Sistema retorna tela de seleção de ponto de ônibus destino
Fluxo alternativo	
1.a 1. Usuário cancela solicitação de parada	2. Sistema retorna cancela operação
3.a 1. Usuário cancela escolha de ponto de ônibus	2. Sistema retorna cancela operação

Tabela 11 – Tabela com caso de uso UC002.

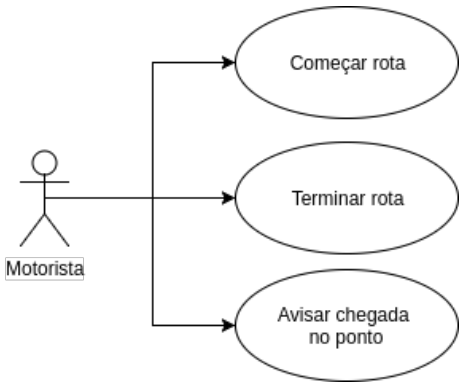

Figura 20 – Diagrama de caso de uso.

3.1.5.3 Narrativas: Casos de Uso

Iniciar uma rota: Este caso de uso acontece quando o motorista inicia sua rota e precisa informar isso ao módulo, para que o dispositivo comece a enviar seus dados de geolocalização para o *web service*.

Terminar uma rota: Este caso de uso acontece quando o motorista termina a rota do ônibus e precisa informar isso ao módulo, para que o dispositivo pare de enviar seus dados de geolocalização para o *web service*.

Avisar chegada no ponto: Este caso de uso acontece quando um deficiente visual solicitou o ônibus e o motorista está parado no ponto de ônibus. Quando motorista avisa que chegou, é disparado um alerta para o deficiente visual.

3.1.5.4 Diagramas de apoio para compreensão funcional

Identificação: UC003	
Nome: Iniciar uma rota	
Atores: Motorista	
Pré-condições: O motorista precisa ter iniciado o módulo	
Pós-condições: -	
Fluxo de eventos	
Ator	**Sistema**
1. Usuário confirma o inicio da rota	2. Módulo começa a enviar dados de geolocalização
	3. Sistema exibe tela com opção de parar envio de dados

Tabela 12 – Tabela com caso de uso UC003.

Identificação: UC004	
Nome: Finalizar uma rota	
Atores: Motorista	
Pré-condições: O motorista precisa ter iniciado uma rota	
Pós-condições: -	
Fluxo de eventos	
Ator	**Sistema**
1. Motorista confirma o fim da rota	2. Módulo para de enviar dados de geolocalização
	3. Sistema exibe tela com opção de iniciar envio de dados

Tabela 13 – Tabela com caso de uso UC004.

Identificação: UC005	
Nome: Avisar chegada no ponto	
Atores: Motorista	
Pré-condições: O motorista precisa ter iniciado uma rota	
Pós-condições: -	
Fluxo de eventos	
Ator	**Sistema**
1. Motorista confirma chegada no ponto de ônibus	2. Módulo enviar aviso de chegada
	3. Sistema exibe tela com opção de parar envio de dados

Tabela 14 – Tabela com caso de uso UC005.

3.2 Módulo do Ponto de Ônibus

3.2.1 Hardware

1. HM-10 - Bluetooth 4.0 BLE module
2. Arduino Uno

3.2.2 Software

- Arduino IDE 1.8.3 ou superior.

3.2.3 Configuração

Para configurar o módulo HM-10 utilizamos o Arduino como ponte. Para realizar tais configurações, foi montado o circuito conforme a figura abaixo.

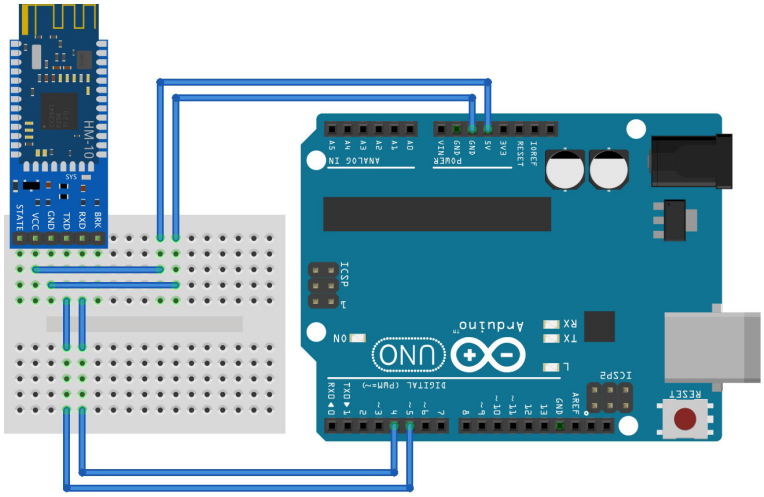

Figura 21 – Módulo Bluetooth HM10.

Após conectar o arduino ao computador, foi utilizado sua *IDE* para escrever o código, que está no apêndice A. No código do Arduino, foi estabelecido uma comunicação serial com o módulo HM-10 para enviar os comandos AT necessários. Esses comandos são para otimizar o uso da bateria e ativar a função *Beacon* do módulo. A seguir, a descrição dos comandos.

Código	Descrição
AT+RENEW	Coloca nos padrões de fábrica
AT+RESET	Reinicia para aplicar os padrões de fábrica
AT+MARJ0xNNNN	Define o valor Marjor
AT+MINO0xNNNN	Define o valor Minor
AT+NAMEMeuBeacon	Define o nome do dispositivo
AT+ADVI5	Define tempo de envio. 5 = 546.25 millisegundos
AT+ADTY3	Define como não pareável
AT+IBEA1	Habilita como Beacon
AT+DELO2	Configura para apenas emitir sinal
AT+PWRM0	Habilita função auto-sleep para economizar energia
AT+RESET	Reinicia para aplicar as configurações

Após configurado, pode ser ligado em uma bateria 3v para utilização.

3.2.4 Referências

HM-10 Bluetooth 4.0 BLE module Datasheet

Arduino IDE

Repositório da Metractive - Como construir Beacons

3.3 Módulo do Ônibus

3.3.1 Hardware

- Intel Edison
- Raspberry Pi 3
- Tela LCD 7"(em breve)
- NEO u-blox 6 GPS Modules

3.3.1.1 Intel Edison

Inicialmente foi adotado o Intel Edison com placa de expansão arduino. Foi escolhido devido a fácil acesso a um exemplar e ótimo hardware. Ele conta com WiFi, Bluetooth, portas

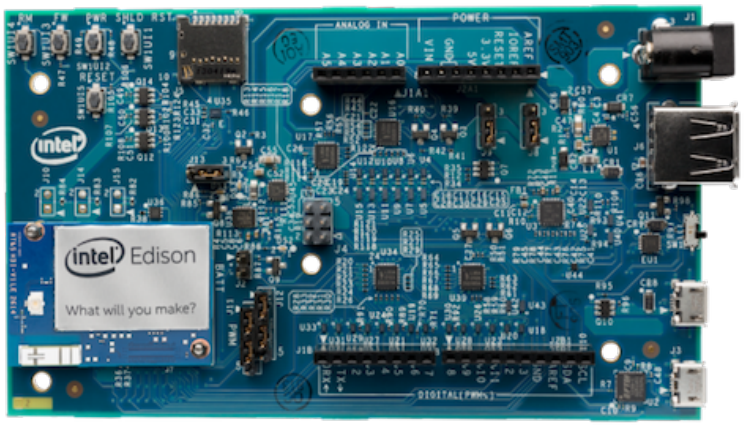

Figura 22 – Intel Edison.

I/O, processador Intel Atom de 500 MHz, 1GB de memória RAM DDR3 e 4GB eMMC.
Sua utilização foi fácil e não obtivemos nenhuma dificuldade em instalar o sistema que escolhemos.

Problemas encontrados em adotar como solução:

Preço

Embora tenha um ótimo hardware e uma empresa séria por trás da sua construção, o preço, em 07/2017, que gira em torno de R$ 600,00, não justifica sua adoção como a melhor solução para o projeto já que existem alternativas com preços melhores e bom desempenho, como o Raspberry Pi 3.

Ausência de controlador gráfico

Este projeto foi elaborado pensando em fazer alertas por meio de um display gráfico. A placa Intel Edison nos permite fazer alertas visuais utilizando LEDs ou pequenos displays OLED, o que não satisfaz a exigência do projeto.

Descontinuidade da placa pela Intel

em 07/2017, a Intel anunciou a descontinuidade do desenvolvimento de algumas placas que fábrica. O Intel Edison foi uma delas.

3.3.1.2 Raspberry Pi 3

Testes realizados no Raspberry Pi 3 demonstraram ser uma boa alternativa ao Intel Edison. Foi fácil a instalação do sistema Android Things e a placa vem com saída HDMI que permite utilizar telas LCD para fazer os alertas visuais. Seu preço, em 08/2017, gira em torno de R$ 150,00, 1/4 do preço do Intel Edison. Seu hardware contém boas especificações.

Embora tenha um hardware com especificações superiores ao Intel Edison, não houve ganho de desempenho ao rodar o sistema, devido a ausência de algoritmos complexos no sistema. Assim, a grande vantagem de se utilizar o Raspberry Pi 3 ao invés do Intel Edison, é seu baixo custo e recurso de chip gráfico.

3.3.1.3 Módulo NEO u-blox 6 GPS

Para realizar o rastreamento do ônibus foi adotado o módulo NEO u-blox 6 GPS Modules, devido a compatibilidade com as placas que contém o sistema embarcado e preço acessível, que gira em torno de R$ 60,00 em 08/2017.

FALAR SOBRE DRIVERS NMEA NO ANDROID THINGS

3.3.2 Software

3.3.2.1 Android Things

Ele é, atualmente, uma versão do Android O reduzida. Sua escolha foi devido a facilidade de embarcar em placas como o Raspberry Pi e Intel Edison, além da variedade de recursos que já estão disponíveis no SO que facilitam o desenvolvimento do módulo, como serviços de geolocalização.

Para obter uma imagem do Sistema Operacional, foi preciso realizar um cadastro na plataforma *Android Things Console* <https://partner.android.com/things/console/>. Foi necesssário cadastrar um produto, neste caso chamamos de *BeaconBusTracker*, conseguindo acesso ao download da imagem do SO.

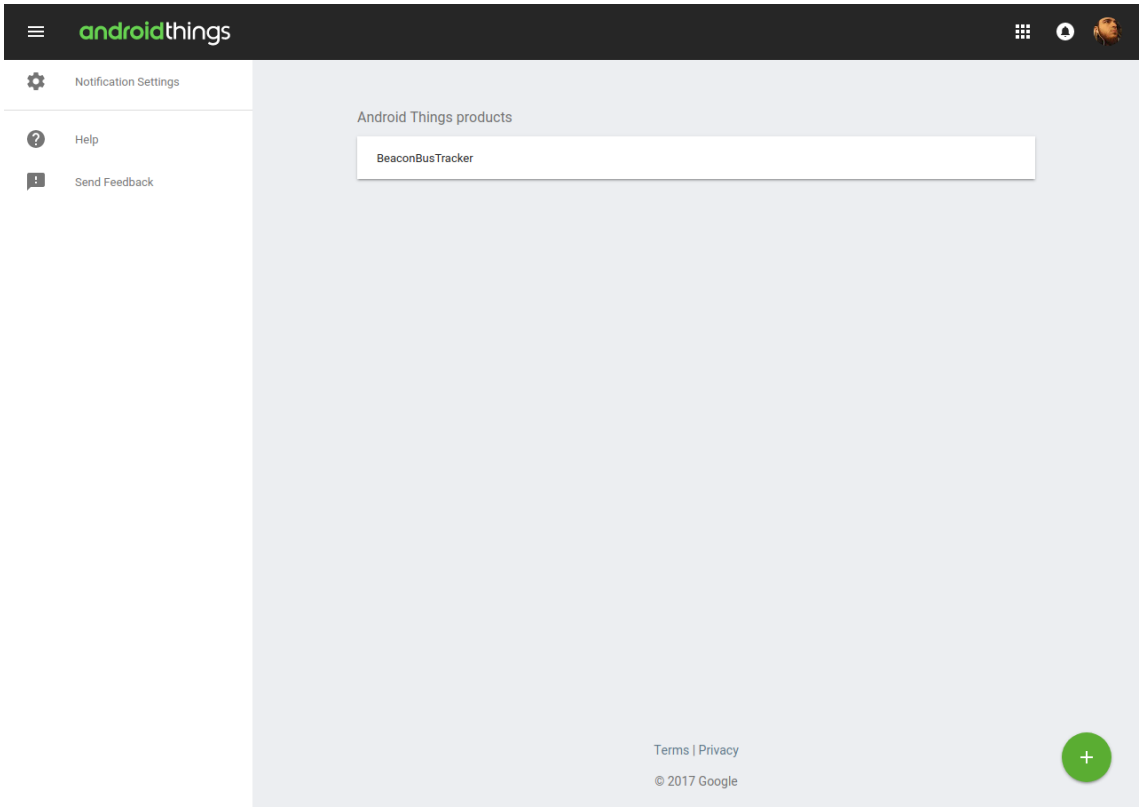

Figura 23 – Página principal do Android Things Console com produto criado.

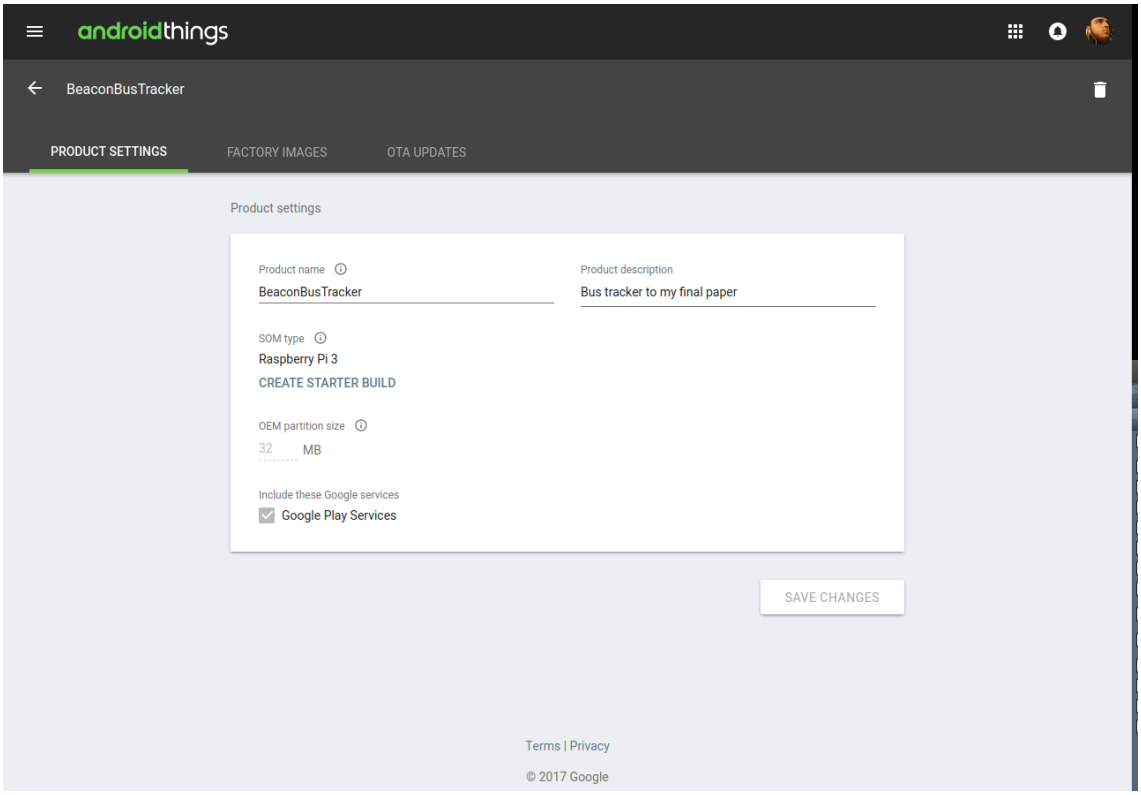

Figura 24 – Página de configurações do produto criado.

LocationManager

O LocationManager é um serviço nativo que observa constantemente os dados de geolocalização do dispositivo. Com ele é possível ser notificado sobre as mudanças da posição, podendo configurar ser notificado quando entra em determinada região, a cada tempo determinado ou quando se move por um distância.

Este serviço abstrai toda lógica de acesso ao hardware e interpretação dos dados. Neste trabalho foi configurado para que ele notifique a geolocalização a cada 10 segundos.

Peripheral Driver Library

Não disponível na versão *Android* para *smartphones*, esta biblioteca oferece suporte no *Android Things* para que possa adicionar sensores e atuadores a placa utilizada, neste caso o *Raspberry Pi 3*. Neste trabalho foi utilizado o *Driver GPS* desta biblioteca para poder configurar o módulo *NEO 6M*. O que possibilita o *LocationManager* obter os dados de geolocalização, já que não existe sensor *GPS* no *Raspberry Pi 3*.

3.3.2.2 IDE

Foi escolhido o Android Studio como IDE do projeto. Ela é desenvolvida pela IntelliJ e tida pelo Google como ferramenta oficial de desenvolvimento para aplicativos Android.

3.3.2.3 Arquitetura

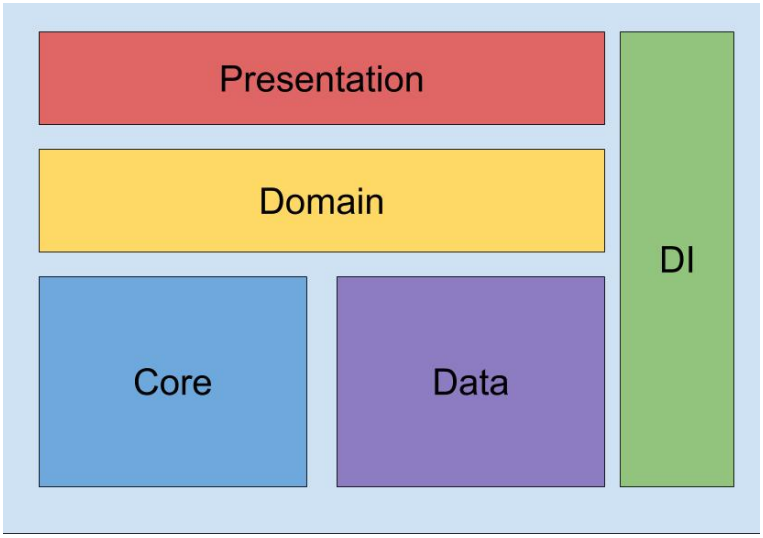

Figura 25 – Diagrama de bloco do módulo do ônibus.

Para desenvolvimento do software, foi adotado o padrão *Clean Archtecture*. Este padrão foi proposto por Robert Ceil Martin, conhecido como Uncle Bob, em 2012. Este padrão de arquitetura procura focar no domínio da aplicação, fazendo frameworks e drivers ser detalhes de implementação. Seguindo seus princípios, a estrutura de camadas ficou da seguinte forma:

Core Não contém nenhuma lógica de negócio. Esta camada provê informações comuns, como configurações estáticas da placa, a toda a aplicação. Possui também classes e interfaces bases.

Data Responsável por prover dados para toda aplicação. Ela adota o Padrão de Arquitetura *Repository*, tendo uma interface de acesso aos dados. Uma grande vantagem em utilizar essa camada com esse padrão de arquitetura, é o respeito a responsabilidade única, um dos princípios do *SOLID*. Ela encapsula toda lógica de busca de dados, assim, caso uma classe precise de algum dado específico, ela solicita através da interface de comunicação e a classe que implementa a interface, cuida de toda lógica de busca de dado, seja um dado

armazenado localmente, em cache ou em um servidor remoto. Tudo fica transparente para a classe que solicitou o dado.

Domain Esta camada encapsula toda regra de negócios da aplicação. Toda vez que é necessário realizar processamentos em dados para satisfazer funcionalidades, é feito por esta camada.

Presentation Responsável por toda interface gráfica. Toda lógica de criação de telas e interceptação de interações do usuário com o aplicativo, é feito aqui. Quando é necessário procurar dados para exibir ao usuário, é feito solicitações para a camada *Domain* ou *Data* para que possa exibi-los.

3.3.2.4 Animações

Um dos tópicos mais presentes sobre melhorar experiência do usuário em *Smartphones*, são as animações. Elas deixam o uso mais fluído e agradável para o usuário. [pesquisar na literatura e colocar aqui]

O *Android Things* provê uma *API* para animações que herdou da versão do *Android* de smartphones. Ela foi utilizada para melhorar a experiência de uso dos motoristas com o módulo, porém, foi observado uma baixa qualidade nas animações. O que fez ter o efeito contrário, pois passa a impressão de ser um sistema de baixo desempenho.

3.3.2.5 Referências

Site Oficial Intel Edison

Datasheet Intel Edison

Anúncio do fim da produção do Intel Edison

Site Oficial Raspberry Pi

Datasheet Raspberry Pi 3

Datasheet NEO u-blox 6 GPS Modules

Site Oficial Android Things

Configuração do Android Things no Intel Edison

Configuração do Android Things no Raspberry Pi 3

3.4 Aplicativo

3.4.1 Telas

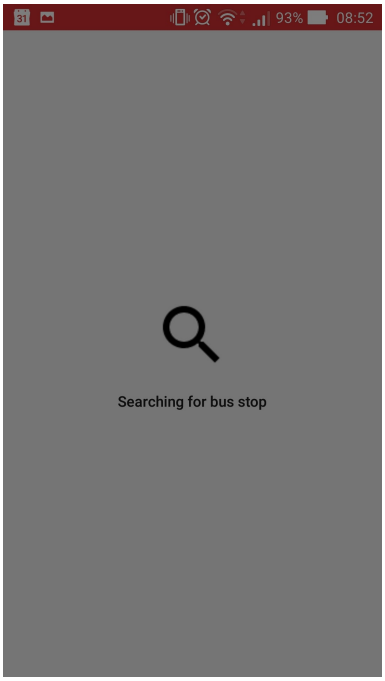

Figura 26 – Tela de busca por um ponto de ônibus do aplicativo móvel.

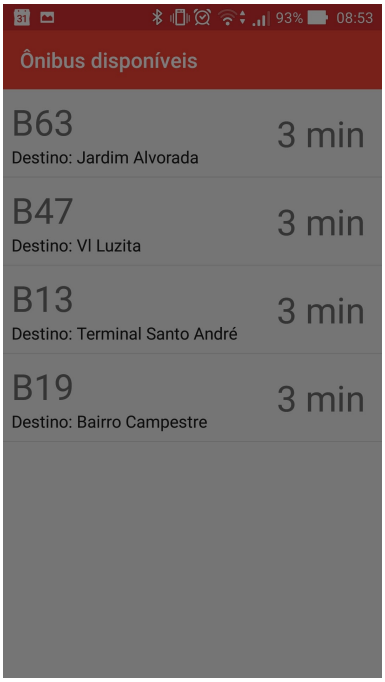

Figura 27 – Tela com lista de ônibus disponíveis do aplicativo móvel.

3.4.2 IDE

Foi escolhido o Android Studio como IDE do projeto. Ela é desenvolvida pela IntelliJ e tida pelo Google como ferramenta oficial de desenvolvimento para aplicativos Android.

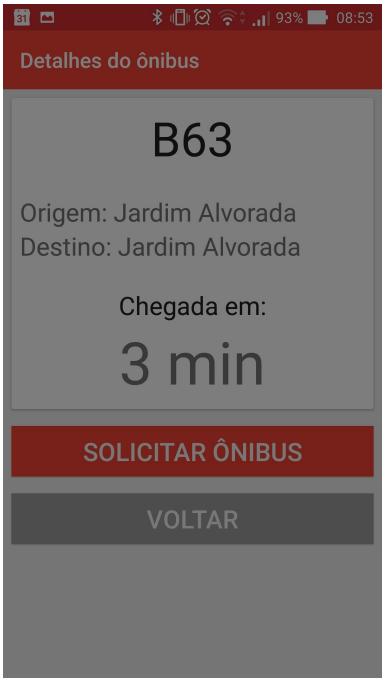

Figura 28 – Tela com detalhes do ônibus do aplicativo móvel.

3.4.3 Arquitetura

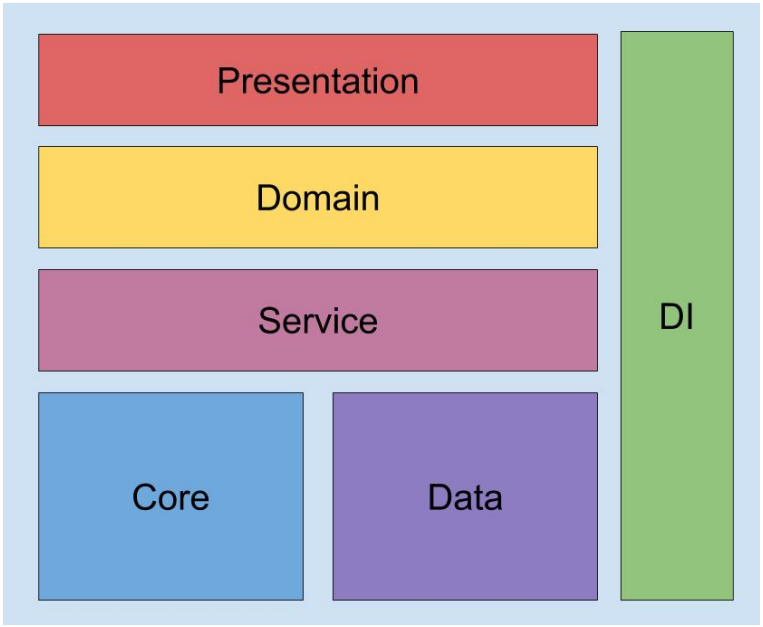

Figura 29 – Diagrama de blocos do aplicativo móvel.

Para desenvolvimento do software, foi adotado o padrão *Clean Architecture*. É um padrão que visa um maior desacoplamento das classes e distruibui bem as responsabilidades.

Core Não contém nenhuma lógica de negócio. Esta camada provê informações comuns, como configurações estáticas da placa a toda a aplicação. Possui também algumas classes e interfaces bases.

Data Responsável por prover dados para toda aplicação. Ela adota o Padrão de Arquitetura *Repository*, tendo uma interface de acesso aos dados. Uma grande vantagem em utilizar essa camada com esse padrão de arquitetura, é o respeito a responsabilidade única, um dos princípios do *SOLID*. Ela encapsula toda lógica de busca de dados, assim, caso uma classe precise de algum dado específico, ela solicita através da interface de comunicação e a classe que implementa a interface, cuida de toda lógica de busca de dado, seja um dado armazenado localmente, em cache ou em um servidor remoto. Tudo fica transparente para a classe que solicitou o dado.

Service Provê serviços para qualquer camada. No caso do aplicativo, a implementação do serviço de voz fica neste pacote e é injeta pelo pacote de Injeção de Dependências.

Domain Esta camada encapsula toda regra de negócios da aplicação. Toda vez que é necessário realizar processamentos em dados para satisfazer funcionalidades, é feito por esta camada.

Presentation Responsável por toda interface gráfica. Toda lógica de criação de telas e interceptação de interações do usuário com o aplicativo, é feito aqui. Quando é necessário procurar dados para exibir ao usuário, é feito solicitações deles para a camada Domain ou Data para que seja possa exibir os dados.

DI Este projeto utiliza o padrão de arquitetura *Injeção de Dependências*. Esta camada provê todas dependências, fazendo a implementação mais limpas nas outras classes, já que não precisam saber como instânciar uma classe, apenas usam.

3.4.4 Áudio Descrição

Uma das funcionalidades do aplicativo é descrição da tela que o deficiente está. O *TalkBack* fala para o usuário em qual componente ele está tocando, porém, não descreve em qual tela ele acabou de entrar. A implementação por áudio descrição foi simples com uso da API nativa *TextToSpeech*, onde podemos passar textos personalizados e o serviço se encarrega de sintetizar a voz.

O uso de uma camada de DI (Injeção de Dependências) facilitou o processo de implementação, fazendo ela na camada de serviço e configurando a instanciação no padrão *Singleton* para que todos que vão utilizar (nesse caso são os *presenters*), apenas solicitem a instância sendo passada por construtor.

3.4.5 Usabilidade

É uma boa prática no desenvolvimento de softwares, sempre confirmar se o usuário tem certeza que deseja executar alguma alteração que possa ter algum impacto no sistema ou em alguma funcionalidade, normalmente lançando alertas para garantir que o usuário não clicou sem querer em algum determinado botão, por exemplo.

Inicialmente foi pensado em usar um *dialog*, um pequeno quadrado que surgi acima de todos componentes da tela com uma mensagem, para que o usuário confirme a ação de adicionar um ônibus como favorito, na tela de confirmação de acompanhamento. Ao testar a aplicação funcionando com *Talkback*, foi observado que o sistema descreve o botão com o seguinte texto: *"Adicionar aos favoritos. Botão, para acionar toque duas vezes"*. Esse texto já faz o usuário se assegurar da sua ação, tornando a prática de lançar um outro alerta ser algo desnecessário, fazendo o usuário ter um trabalho a mais de deslizar o dedo pela tela para encontrar os botões de *OK* e cancelar do *dialog*.

Com base nessas observações, não foi implementado *dialogs* de confirmação. Deixando a responsabilidade de afirmar as ações do usuário para o *Talkback* fazer. Embora seja uma pequena ação, tem grande impacto na usabilidade no quesito facilidade de uso do aplicativo por parte do usuário final, que são os deficientes visuais.

Outro ponto observado durante os testes foi a rotação da tela. O sistema operacional Android permite que aplicativos implementem variações da tela de acordo com a orientação do dispositivo, retrato ou paisagem. Isso permite que o layout do aplicativo se adapte a nova disposição de espaço.

Pensando no usuário final, para saber onde está cada elemento, ele precisa deslizar o dedo pela tela para conhecer a localização de cada um. Se ao rotacionar o aparelho, a disposição dos elementos mudar, o usuário precisa verificar novamente onde está cada um.

Como este trabalho desenvolve um aplicativo com poucos elementos na tela para simplificar o uso por deficientes visuais, foi bloqueado a mudança de tela ao rotacionar o aparelho.

Isso garante um melhor conforto ao usar o aplicativo.

3.5 Web service

3.5.1 Provedor de servidor

Para tornar a aplicação acessível ao público, torna-se necessário dispor da utilização de um servidor que fique disponível constantemente.
Foram pesquisados três provedores de servidores, analisando qual deles apresentava melhores condições para realização do *deploy* da aplicação. Foram eles a GoDaddy, Umbler e LocaWeb.
Primeiro fator, que fez o serviço da GoDaddy ser excluído da lista, foi a questão de período de teste. A GoDaddy possui um período nulo de teste do serviço. Já a Umbler e LocaWeb fornecem créditos para novos desenvolvedores.
Em seguida, foi observada a disponibilidade de informações sobre como realizar o *deploy* de um servidor em Node.js. Nesse aspecto, a LocaWeb não demonstra com clareza os procedimentos a serem seguidos.
A Umbler acabou sendo selecionada por ser muito intuitiva na hora de selecionar a linguagem da aplicação e qual o banco de dados a ser utilizado, com poucos cliques é possível iniciar um servidor funcional.

3.5.2 Arquitetura

Devido o uso do Express.js como intermediário para carregar os *middlewares*, a arquitetura utilizada para o desenvolvimento da aplicação segue o padrão MVC.

Node Cuida da parte de escalabilidade e conexão e escalabilidade do servidor com os usuários da aplicação.

MongoDB Representa o banco de dados da aplicação, com as collections necessárias para o funcionamento da aplicação.

O Node, está subdividido da seguinte forma:

App Conjunto com todas as funções da aplicação.

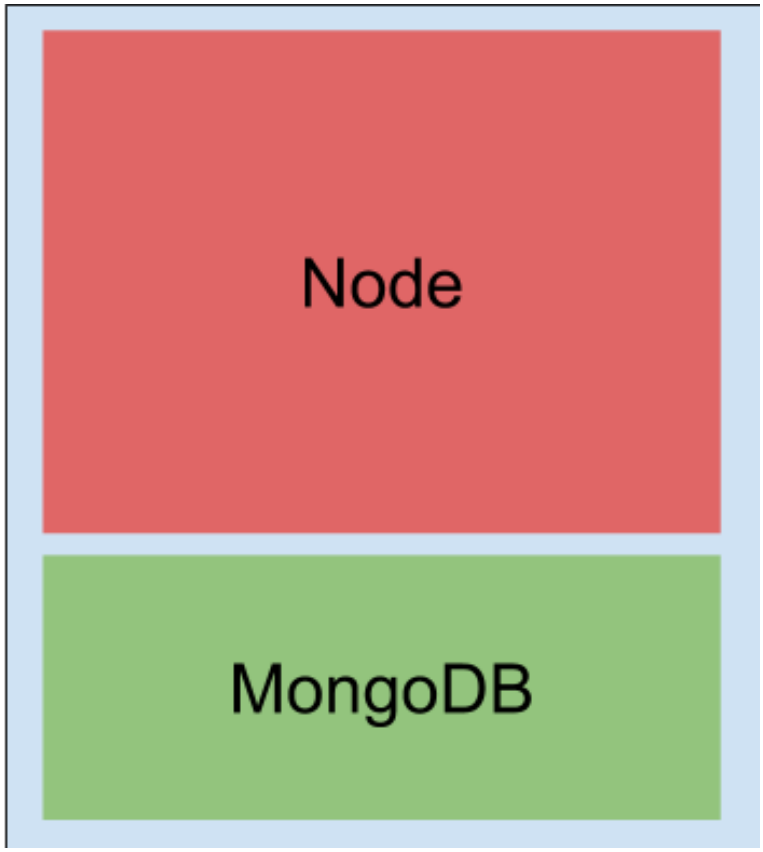

Figura 30 – Diagrama de blocos do webservice.

Config Possui os arquivos necessários para realizar as conexões com o servidor, banco de dados e o sistema de notificações.

Dentro do app é onde se observa a utilização da arquitetura MVC, representada da seguinte forma:

Models Possui as regras de negócio da aplicação, com as informações necessárias para a criação das *collections* do MongoDB através do mongoose e os arquivos com as funções de previsão de tempo e de correção de posição.

Controllers Possui as funções que utilizam os dados do banco de dados para gerar as informações adequadas aos usuários.

Routes Possui os arquivos com as rotas dos *endpoints*, indicando qual arquivo e função dos *controllers* será utilizado para a operação necessária pelo usuário, seguindo o padrão REST.

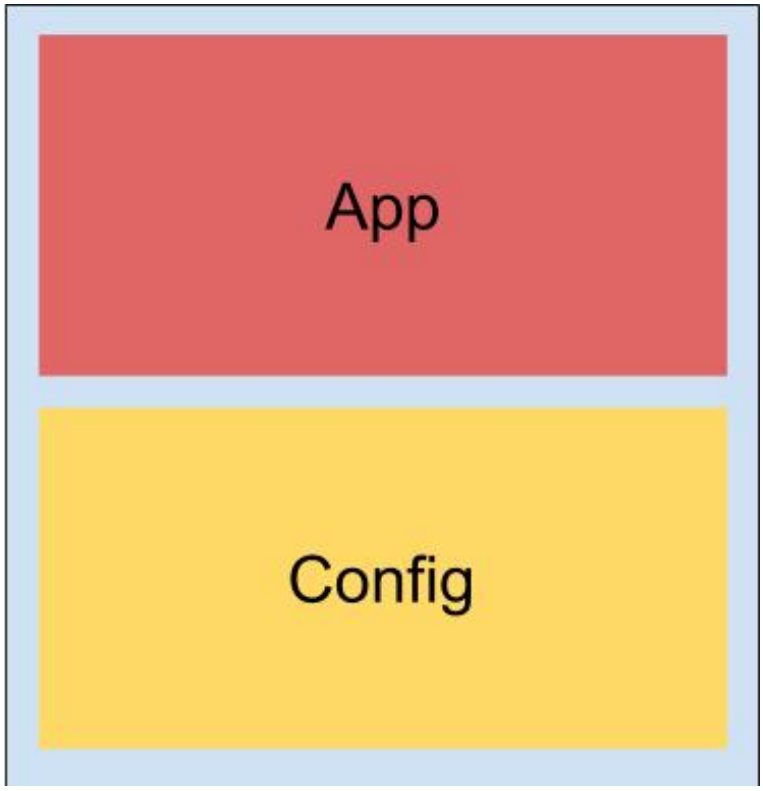

Figura 31 – Diagrama de blocos do Node.

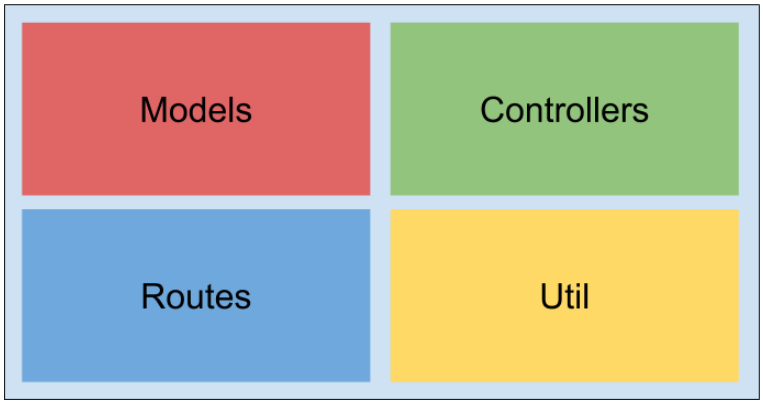

Figura 32 – Diagrama de blocos do diretório app

Util Possui as fórmulas necessárias para o desenvolvimento de certas funções da aplicação.

3.5.3 Localização do ônibus

Ter conhecimento da posição do ônibus se faz necessário para realizar a previsão de chegada em um ponto de parada. Neste trabalho, sempre que o *Web Service* recebe as coordenadas geográficas do ônibus, elas são corrigidas para um ponto válido da rota que este está fazendo.

Quando o *Web Service* recupera as informações da rota do veículo, ele obtém todos os pontos referentes a ela, onde cada um tem informações de latitude, longitude e se este é um ponto de parada. Então é verificado qual ponto está mais próximo a localização recebida, o mais próximo é associado ao ônibus.

O cálculo de distância é feito a partir da seguinte fórmula:

FORMULA AQUI

Busca Binária

"Binary search is to algorithms what a wheel is to mechanics: It is simple, elegant, and immensely important." Udi Manber, Introduction to Algorithms

Para otimizar a busca pelo ponto mais apto, é realizado uma busca binária. Ela segue o paradigma de dividir para conquistar. Tendo como premissa um *array* de dados ordenados, primeiro observa os extremos do array, para em seguida observar o ponto médio entre eles. Caso um desses pontos seja igual ao valor procurado, assume ele como solução. Caso contrário se observa qual sub *array* é mais apto, início até o centro ou do centro até o fim. O mais apto é submetido a uma nova busca binária.

A vantagem em se utilizar uma busca binária ao invés de uma busca sequencial pode ser observado pelas suas complexidade. A busca binária possui complexidade $O(\log_2 n)$, enquanto a sequencial possui complexidade $O(n)$

A tabela a seguir mostra o número de vezes que cada busca iria precisar fazer para chegar no ponto mais apto.

Rota	Número de Pontos	Busca Binária ($O(\log_2 n)$)	Busca Sequencial ($O(n)$)
1	410	$\simeq$9	410
2	500	$\simeq$9	500
3	675	$\simeq$10	675
4	708	$\simeq$10	708
5	725	$\simeq$10	725

A implementação da busca ficou da seguinte forma:

Primeiro é calculado a distância entre a localização recebida do ônibus e o primeiro ponto da rota. Em seguida, calculamos a distância entre a localização recebida e o ultimo ponto

da rota. Com isso temos a distânciaInicio e distânciaFim.

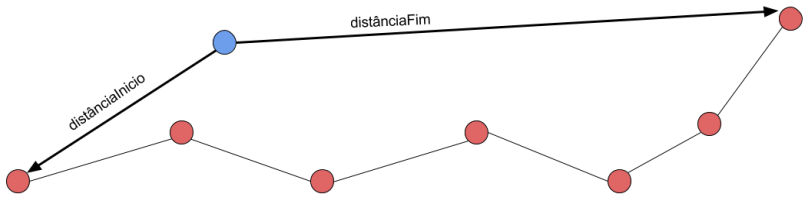

Figura 33 – Representação da busca binária.

Um terceiro cálculo é feito, a distância entre a localização recebida e o ponto do meio da rota.

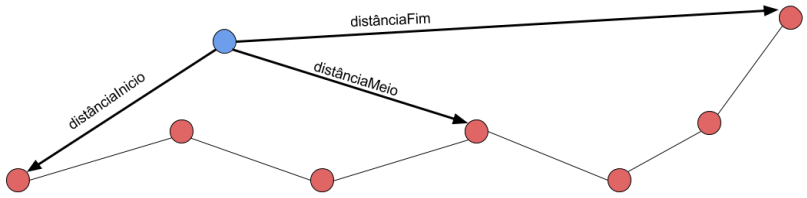

Figura 34 – Representação da busca binária.

O algoritmo então verifica qual metade é mais apta, baseando-se nas distâncias mais curtas. Se distânciaFim é maior que distânciaInicio, é feito uma recursividade assumindo distânciaMeio como fim.

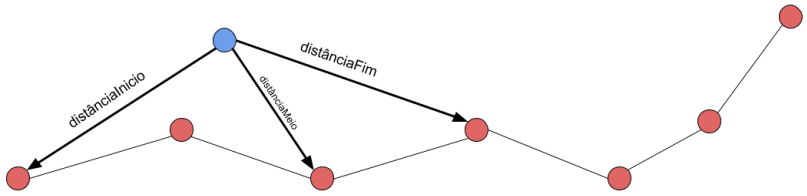

Figura 35 – Representação da busca binária.

Fazendo isso sucessivamente, chega uma hora que não é mais possível dividir, então o ponto mais próximo assume o posto de mais apto e é associado ao ônibus.

3.5.4 Previsão de chegada

Para calcular a previsão de chegada de um ônibus, precisamos saber a ultima localização conhecida dele e o ponto de parada escolhido. Com essas duas informações podemos recuperar todos dados necessários para o cálculo.

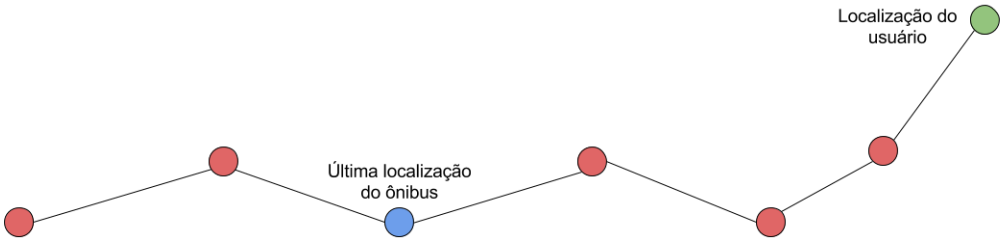

Figura 36 – Representação de uma rota.

Primeiro é recuperado todos os dados da rota que o ônibus está fazendo. Cada rota possui N pontos associados a ela, onde cada um tem a informação de latitude, longitude e se este é um ponto de parada. Sabendo em qual ponto da rota o ônibus está e qual ponto de parada o usuário se encontra, podemos pegar todos os pontos que estão entre eles e calcular a distância com a seguinte fórmula.

$$t = \frac{s}{v}$$

A partir disso somamos a distância dentre todos esses pontos, com isso dividimos pela velocidade da pista e sabemos quanto tempo aproximado falta para o ônibus chegar na parada.

3.5.5 Alertar motorista sobre parar

O *Android Things* não suporta serviço de *push notification*. Para enviar mensagens instantâneas ao módulo foi utilizado o *Realtime Database*, do *Firebase*, que permite observar um nó. Sempre que existe uma alteração deste, quem está observando é notificado.

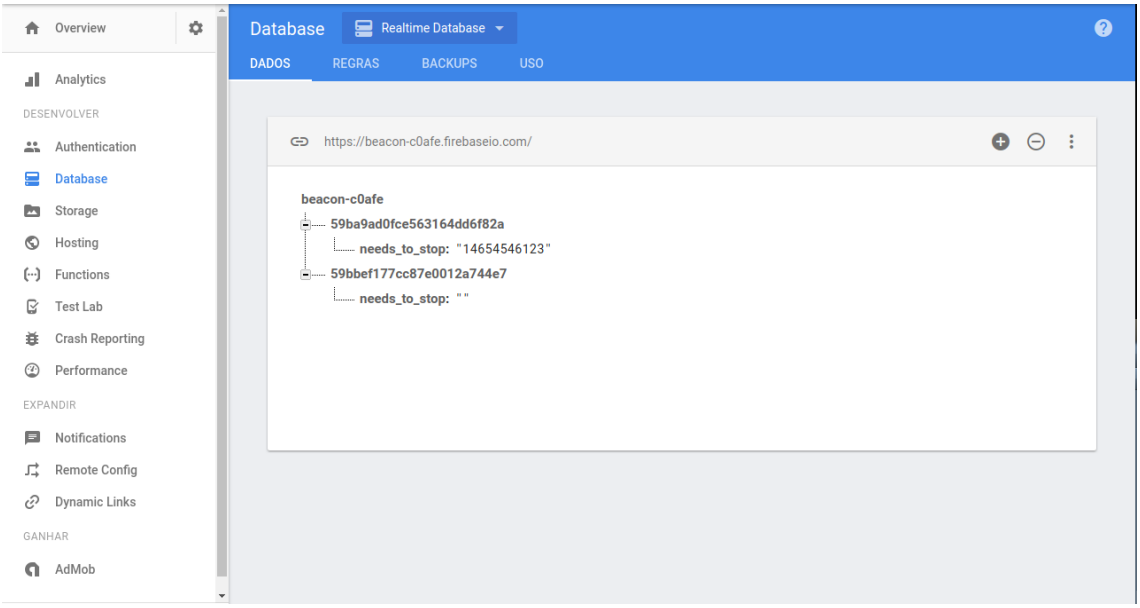

Figura 37 – Representação de uma rota.

Quando um módulo do ônibus é conectado a internet, ele cria um nó com seu ID na base de dados e um nó filho intitulado *needs_to_stop*, e passa a observá-lo. Quando o valor do nó filho é alterado, o módulo então dispara uma notificação para o motorista sobre a necessidade de parar no próximo ponto de ônibus.

O responsável por alterar o valor do nó filho é o *Web Service*. Como visto na seção *Localização do Ônibus*, é feito uma varredura para descobrir o ponto da rota que o ônibus se encontra. Em seguida também é verificado qual o próximo ponto que ele vai passar. Caso o próximo ponto esteja na lista de requisições de parada, é então alterado o valor do nó filho para que aquele módulo alerte o motorista.

3.5.6 Alertar usuário que ônibus chegou

Quando o motorista chega na parada, ele pressiona um botão para anunciar sua chegada. Quando *Web Service* recebe esse alerta por meio de uma requisição *HTTP*, ele verifica quais usuários estão aguardando aquele ônibus, então é disparado uma *push notification*, por meio do *Firebase*, avisando que o ônibus que ele aguarda está parado.