UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

FRANCISCO LOPES DANIEL FILHO

UMA API PARA DETECÇÃO DE PRESENÇA DE DISPOSITIVOS MÓVEIS EM AMBIENTES INDOOR

QUIXADÁ – CEARÁ

2016

FRANCISCO LOPES DANIEL FILHO

UMA API PARA DETECÇÃO DE PRESENÇA DE DISPOSITIVOS MÓVEIS EM AMBIENTES INDOOR

Monografia apresentada no curso de Sistemas de Informação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Sistemas de Informação. Área de concentração: Computação.

Orientador: Prof. Dr. Jefferson de Carvalho Silva

QUIXADÁ – CEARÁ

2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

D185a Daniel Filho, Francisco Lopes.
Uma API para detecção de presença de dispositivos móveis em ambientes indoor / Francisco Lopes Daniel Filho. – 2016.
52 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2016.
Orientação: Prof. Dr. Jefferson de Carvalho Silva.

1. Interface de programas aplicativos (Software). 2. Android (Programa de computador). 3. Internet sem fio. I. Título.

CDD 005

FRANCISCO LOPES DANIEL FILHO

UMA API PARA DETECÇÃO DE PRESENÇA DE DISPOSITIVOS MÓVEIS EM AMBIENTES INDOOR

Monografia apresentada no curso de Sistemas de Informação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Sistemas de Informação. Área de concentração: Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Jefferson de Carvalho Silva (Orientador)
Campus Quixadá
Universidade Federal do Ceará – UFC

Lucas Ismaily Bezerra Freitas
Campus Quixadá
Universidade Federal do Ceará - UFC

Francisco Helder Candido dos Santos Filho
Campus Quixadá
Universidade Federal do Ceará - UFC

À minha mãe e minhas irmãs.

AGRADECIMENTOS

Ao meu orientador, Professor Doutor Jefferson de Carvalho Silva, que sempre teve disponibilidade para me ouvir, orientar e interesse em guiar-me durante todo o projeto.

À minha tia Salete Daniel, e ao meu tio Carlos Apolônio que sempre me motivaram e me deram incentivo para seguir em frente.

Dedico esse trabalho a todos os meus amigos, Kerley Dantas, Maxdyllon Rodrigues, Paulo Rennê, Alan Martins, Leonel Júnior, Araújo Filho, Danrley Teixeira, Anderson Lemos, Flávio Barros, Lucas Henrique, Amanda Oliveira, Alysson Gomes, Yago Alves, Allan Vidal, Matheus Pereira, Andreza Brito, Cainã Mello, e todos os demais, por fazer de minha vida acadêmica uma experiência inesquecível, e por me apoiarem nos momentos mais difíceis.

Aos professores Davi Romero e Lucas Ismaily por todos os conselhos dados e aos demais professores, que de forma direta ou indireta, me passaram conhecimentos e me tornaram uma pessoa mais madura intelectualmente.

“Somos todos feitos do mesmo pó de estrelas.”

(Carl Sagan)

RESUMO

As pessoas passam a maior parte do seus tempos em ambientes fechados (*indoor*). Nesse cenário, algumas aplicações podem querer localizar usuários dentro de estabelecimentos. Infelizmente, a tecnologia GPS não possui uma boa performance em ambientes indoor, então foram desenvolvidas técnicas de posicionamento usando o sinal WiFi. WiFi é um meio alternativo para localização indoor, e o mesmo possui a vantagem de não necessitar de uma infraestrutura específica. O principal objetivo deste documento é apresentar uma API para a plataforma de desenvolvimento Android, visando ajudar outros desenvolvedores na criação de aplicações para ambientes indoor usando o sinal WiFi.

Palavras-chave: API. Android. WiFi.

ABSTRACT

People spend about most of their time indoors. In this scenario, some applications may want to locate users inside buildings. Unfortunately, GPS technology does not have a good indoor performance, so we developed positioning techniques using WiFi signal. WiFi is an alternative way for indoor location and has the advantage of not requiring expensive specific infrastructure. The main goal of this document is to present an API for Android development platform aiming to help other developers in creating applications for indoor environments, using WiFi signal. To validate the API, we present a small test case application, using a client-server approach.

Keywords: API. Android. Wireless

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous Javascript and XML
AP	Access Point
API	Application Programming Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CSV	Comma-separated Values
GPS	Global Positioning System
GPU	Graphics Processing Unit
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
ISM	Industrial, Scientific and Medical
JSON	Javascript Object Notation
LBS	Location Based Services
LCS	Location Services
MAC	Medium Access Control
MVC	Model View Controller
NoSQL	Not Only SQL
ORM	Object Relational Mapping
RFID	Radio-Frequenci Identification
RSSI	Received Signal Strength Indicator
SSID	Service Set Identifier
TI	Tecnologia da Informação

UFC	Universidade Federal do Ceará
WiFi	Wireless Fidelity
WLAN	Wireless Local Area Network
XML	Extensive Markup Language

SUMÁRIO

1 INTRODUÇÃO

Com o surgimento dos computadores, houve uma mudança considerável no modo de armazenamento, controle e processamento de informações. Empresas adotaram o uso de computadores para automatizar seus processos, como controle de custos, funcionários, produtos e até suporte de informações estratégicas para tomada de decisões. O avanço da tecnologia levou os computadores a se tornarem cada vez menores e com maior poder de processamento, passando a fazer parte não apenas do âmbito empresarial mas também de ambientes domésticos.

Tendo dispositivos tão pequenos que cabem na palma da mão, as pessoas passaram a ter acesso a informações independentemente de suas localizações podendo, inclusive, estarem em movimento. Essa comunicação entre dispositivos, em que não há a necessidade dos mesmos estarem fixos em um local, deu origem ao conceito de computação móvel. A computação móvel permite que usuários tenham acesso a serviços independentemente de sua localização, tendo acesso a informação em qualquer lugar e a qualquer momento (FIGUEIREDO; NAKAMURA, 2003).

Os primeiros telefones celulares possuiam funções limitadas a receber e efetuar ligações. Com o surgimento do *smartphone*, essas funções foram expandidas a acessar internet, escutar música, ler livros, tirar fotos, dentre outras. Além disso, um *smartphone* pode oferecer ao seu usuário informações sobre a sua velocidade, temperatura ambiente, orientação e até mesmo a localização geográfica, todas obtidas através de sensores contidos no mesmo.

Aliado à computação móvel, houve um aumento significativo no uso de dispositivos móveis. Segundo a 27º Pesquisa Anual do Uso de TI realizada em 2016, no Brasil há 300 milhões de dispositivos móveis *wireless* conectáveis à internet sendo que 71% são *smartphones* (MEIRELLES, 2016).

Analisando o crescente uso de *smartphones* e as informações oferecidas pelo mesmo, surgiram diferentes serviços que utilizam a localização geográfica como principal informação. Segundo Virrantaus et al. (2001) esses serviços são categorizados em *LoCation Services* (LCS) e *Location Based Services* (LBS). A diferença entre LCS e LBS se dá pelo fato que o objetivo final da LCS é informar a localização geográfica do usuário enquanto a LBS utiliza a localização geográfica do dispositivo para disponibilizar informações ao usuário.

Um exemplo de um serviço LBS seria uma pessoa procurando um restaurante a partir de uma consulta em seu *smartphone*, informando o tipo de comida, preço e distância até o mesmo. Ao final da consulta, um mapa é apresentado na tela do dispositivo contendo uma lista

de restaurantes que satisfazem aos requisitos informados.

Um meio para oferecer serviços de localização é com o uso do *Global Positioning System* (GPS), contido na maioria dos *smartphones*. O GPS é um sistema utilizado para se obter o posicionamento de maneira rápida e precisa em qualquer lugar da Terra (MIYABUKURO, 2015). Porém, o mesmo possui limitações quando o usuário do dispositivo se encontra em ambientes fechados (*indoor*) pelo fato que o sinal que permite calcular a posição do dispositivo não chega nas melhores condições (SIMÕES, 2015).

As pessoas passam cerca de 80-90% do seu tempo em ambientes *indoor*. Devido ao problema do sinal de GPS não possuir um bom desempenho em ambientes fechados, foram desenvolvidas técnicas de posicionamento *indoor* utilizando tecnologias *wireless* como infravermelho, RFID, *WiFi*, *Bluetooth*, dentre outras (SIMÕES, 2015). O uso do *WiFi* pode ser um meio alternativo para a localização *indoor* além de dispor da vantagem de não necessitar de infraestrutura específica, uma vez que há cerca de 122 milhões de *hotspots* em todo mundo, confirmando a viabilidade do seu uso (IPASS, 2016).

Os Sistemas de Posicionamento *Indoor* podem ser utilizados em diversas áreas. No âmbito comercial, localizando materiais em estoque e disponibilizando informações sobre produtos para clientes a partir da proximidade de determinada prateleira. No âmbito educacional, detectando a presença de alunos na sala de aula. No âmbito hospitalar, determinando a presença de pacientes ou médicos.

O método mais utilizado para posicionamento *indoor* através das redes IEEE 802.11 é a utilização da indicação da potência do sinal recebido, mais conhecido como RSSI (CONTE et al., 2015).

Visando a problemática apresentada, esse trabalho foca no desenvolvimento de uma API para a plataforma Android. A mesma utilizará algoritmos de posicionamento baseados no indicador de potência do sinal recebido (RSSI) para detecção de presença de dispositivos móveis. Também é almejado a contribuição para a comunidade de desenvolvedores na criação de aplicações móveis em ambientes fechados.

Com o desenvolvimento de uma API que facilite a construção de aplicações móveis em ambientes fechados, se torna possível a elaboração de sistemas do interesse do Campus da UFC em Quixadá. O mesmo conta com cerca de 1000 alunos em seis cursos de gradução na área de tecnologia de informação. Através de um levantamento *in-loco* com 108 estudantes, foi verificado que 97,2% dos estudantes entrevistados possuem algum tipo de dispositivo móvel

conectado à internet.

O objetivo geral deste trabalho é implementar uma API para a plataforma Android que forneça meios de detectar a localização relativa de um dispositivo móvel em um determinado ambiente. Os recursos de *hardware*, no caso os dispositivos de rede *WiFi*, usados pela API serão os diponíveis no Campus da UFC Quixadá.

Os objetivos específicos deste trabalho são:

- Implementar uma aplicação de localização *indoor*, no Campus da UFC em Quixadá, a partir da API proposta. Sendo ela responsável por detectar a presença de um estágiário em seu horário de atividade.
- Fornecer uma documentação completa *on-line* para que terceiros possam criar outras aplicações sobre a API proposta e, até mesmo, extender suas classes adicionando novas funcionalidades. O código está disponível na ferramenta **GitHub**.

2 TRABALHOS RELACIONADOS

Nessa seção serão apresentados alguns trabalhos relacionados destacando as semelhanças e diferenças com o desenvolvido nesse trabalho.

2.0.1 IndoorNav

Em Simões (2015) é apresentada uma aplicação para a plataforma Android que tem o objetivo, localizar alunos, professores e visitantes dentro de um Campus da Universidade de Lisboa. A partir da localização do dispositivo, informações são apresentadas ao seu portador tendo como base a proximidade de determinado local. Caso o portador esteja perto de um refeitório, um menu com todas as refeições do dia pode ser apresentado.

A aplicação desenvolvida por Simões (2015) utiliza a arquitetura Cliente/Servidor. Na parte do servidor, foram mapeados todos os pontos de interesse no edifício, como bibliotecas, salas de aula e gabinetes de professores. Essas informações de mapeamento consistem em coordenadas geográficas da posição dos pontos de acesso, além de uma breve descrição do ponto e o piso onde o mesmo se localiza (SIMÕES, 2015).

Já a aplicação cliente possui uma base de dados local com informações sobre a localização de todos os pontos de acesso espalhados ao longo do edifício, logo o funcionamento não é comprometido caso haja algum problema com o servidor.

Assim como Simões (2015), o trabalho desse documento está objetivado em localizar *smartphones* em ambientes *indoor* fazendo o uso das redes 802.11. Porém o seu trabalho não consiste na contrução de uma *API*, e sim de uma aplicação final voltada para o ambiente do Campus da Universidade de Lisboa, impossibilitando o desenvolvimento de outras aplicações *indoor* por terceiros.

2.0.2 Plataforma de Localização de Dispositivos Móveis

Carvalho (2007) apresenta o desenvolvimento de uma aplicação LBS para localização *indoor* de dispositivos móveis dentro da Universidade de Trás-os-Montes e Alto Douro (UTAD) utilizando redes *WiFi*.

Além de oferecer a localização de dispositivos móveis dentro da rede da UTAD, a aplicação oferece serviços que podiam ser acessados por aplicações de terceiros. Ela foi desenvolvida para ser utilizada em qualquer local. A aplicação utiliza arquitetura Cliente/Servidor

e é no servidor onde se efetua todo o cálculo de posição dos dispositivos móveis.

O trabalho de Carvalho (2007) se diferencia do proposto nesse documento no aspecto de não abordar a contrução de uma API, mas sim de um sistema final, dificultando o aprimoramento e desenvolvimento de novas funcionalidades por terceiros. Outro ponto distinto é o fato da aplicação Cliente ser desenvolvida para aparelhos *desktops* e não para *smartphones*. Além da técnica utilizada para se obter a posição do dispositivo, técnica de triangulação, diferente da técnica utilizada na API, *RSSI-Based*.

3 FUNDAMENTAÇÃO TEÓRICA

3.0.1 WiFi

WiFi se refere a *Wireless Fidelity*. Esse termo é normalmente utilizado para se referir a redes WLAN que operem empregando alguma das especificações IEEE 802.11 (802.11a, 802.11b, 802.11g, 802.11n). As redes 802.11 podem ser usadas para o provimento de um canal de comunicação entre dois terminais, tornando-se uma rede *ad hoc* e possuindo funcionalidade similar ao *Bluetooth*. Porém, um ponto de acesso 802.11 é empregado para prover a comunicação entre diferentes terminais em uma pequena área (CRUZ et al., 2011).

A primeira versão do padrão IEEE 802.11 foi lançada em 1997, e previa taxas de transmissão de 1 a 2 megabits e utilizava a faixa dos 2.4 GHz. Esse padrão levou à criação de produtos com baixa compatibilidade entre si, mas consolidaram a base para o desenvolvimento dos padrões atuais. As redes IEEE 802.11 operam nas bandas de ISM (*Industrial, Scientific and Medical*) e portanto não necessitam de licença para funcionar (CARVALHO, 2007).

Existem outras versões do padrão 802.11, como: 802.11a, 802.11b, 802.11g e 802.11n (CRUZ et al., 2011). A versão 802.11a começou a ser desenvolvida antes da 802.11b porém, foi finalizada poucos dias depois da publicação da mesma. A 802.11a utiliza a faixa de frequencia de 5 GHz e oferece uma velocidade teórica de 54 megabits. O fato de poucos dispositivos utilizarem a frequência de 5 GHz faz com que haja uma menor interferência na transferência de informações.

A versão 802.11b foi publicada em Outubro de 1999, dando um grande impulso na popularização da tecnologia por proporcionar que placas de diferentes fabricantes se tornassem compatíveis. Essa versão opera na banda de 2.4 GHz e suporta uma taxa máxima de 11 Mbps (CRUZ et al., 2011). Em seguida surge o padrão 802.11g que utiliza a mesma faixa de frenquência do 802.11b, 2.4 GHz, levando a intercompatibilidade dos dois padrões. Suporta a taxa máxima de 54 Mbps, mesma taxa que a versão 802.11a.

Já a versão 802.11n foi publicada em 2009 com o objetivo de oferecer velocidades reais de transmissão superiores as redes cabeadas de 100 megabits e para isso foram combinados algoritmos de transmissão e o uso do MIMO (*multiple-input multiple-output*) que permite que a placa utilize diversos fluxos de transmissão, transmitindo os dados de forma paralela (CRUZ et al., 2011).

A Figura 1 apresenta as versões abordadas acima e suas principais diferenças.

Figura 1 – Principais versões das redes 802.11

Protocolo	Frequência	Velocidade (típica)	Velocidade (nominal)	Alcance interno aproximado (dependende de número e tipo de paredes)	Alcance externo aproximado (passando por uma parede)
802.11	2.4 GHz	0.9 Mbps	2 Mbps	20 metros	100 metros
802.11a	5 GHz	23 Mbps	54 Mbps	35 metros	120 metros
802.11b	2.4 GHz	4.3 Mbps	11 Mbps	38 metros	140 metros
802.11g	2.4 GHz	19 Mbps	54 Mbps	38 metros	140 metros
802.11n	2.4 GHz 5 GHz	74 Mbps	248 Mbps	70 metros	250 metros

Fonte: MIYABUKURO (2015)

Para informar a sua presença aos dispositivos que estão ao seu redor, um ponto de acesso utiliza canais de *Broadcast* contendo o seu *timestamp* e o nome da rede ou SSID (*Service Set Indentifier*), todos esses dados são resposáveis pela sua indentificação. Cada ponto de acesso transmite periodicamente quadros chamados de *beacon* (*beacons frames*) contento tais informações (CRUZ et al., 2011). A partir desses quadros os dispositivos que estão ao redor conseguem detectar sua presença, podendo assim, conectar-se ao mesmo.

Quando há muitos obstáculos entre o ponto de acesso e o dispositivo final, pode haver atenuação ou, dependendo do tipo de material do obstáculo, até mesmo uma perda total na ondas emitidas. Além de perder a sua intensidade, as ondas *Wi-Fi* podem sofrer alterações em suas direções (CARVALHO, 2007).

Para se localizar um dispositivo em um ambiente fechado utilizando redes *WiFi*, faz-se necessária uma conversão da força do sinal recebido vindo do ponto de acesso em unidades de distância. Esse sinal, também conhecido por RSSI, informa a potência do sinal recebido pelo dispositivo final. A precisão varia entre 3 a 30 metros usando algoritmos baseados em RSSI (CONTE et al., 2015).

O padrão IEEE 802.11 vem sendo utilizado por Sistemas Indoor por causa de seu baixo custo, já que não se faz necessário a instalação de uma estrutura especifica para tal (CONTE et al., 2015).

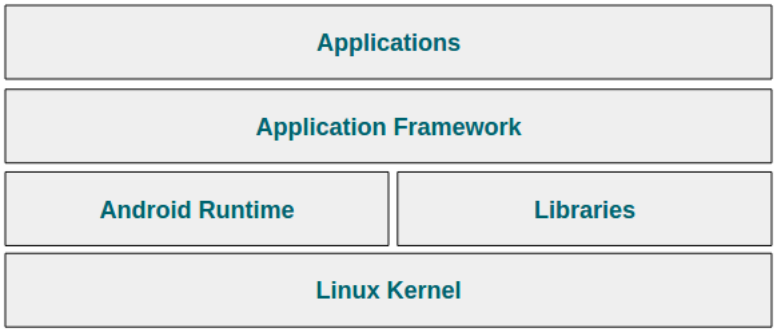
Na API proposta nesse trabalho, foram utilizadas redes *WiFi* do Campus da UFC em Quixadá para se obter uma localização relativa de *smartphones* baseada na transformação do indicador de potência do sinal recebido em unidades de distância.

3.0.2 Android

Desenvolvido pelo Google, Android é uma plataforma de desenvolvimento de aplicações móveis que possui o sistema operacional baseado no kernel 2.6 do Linux, responsável por gerenciar toda a memória, processos, *threads*, segurança de arquivos, pastas, além de gerenciamento de redes e drivers. A linguagem Java é utilizada para desenvolver suas aplicações (LECHETA, 2013).

A Figura 2 apresenta a arquitetura da plataforma Android com suas devidas divisões.

Figura 2 – Arquitetura do Android



Fonte: Adaptado de Brahler (2010)

No topo da arquitetura se encontra a camada *Applications*, responsável por encapsular todas as aplicações básicas, tais como agenda de contatos, galeria de fotos, *browser* para acesso a sites, entre outras. É nessa camada que estão as aplicações desenvolvidas por terceiros (SIMÕES, 2015).

Logo abaixo, na camada *Application Framework* há o gerenciamento dos recursos entre programa e processos. É nessa camada onde a classe *Intent* está localizada. Essa classe envia uma mensagem para o sistema operacional informando a intenção de realizar determinada tarefa, proporcionando assim a integração entre aplicações (LECHETA, 2013).

Na camada inferior se localiza o *Android Runtime*, onde todas as aplicações são escritas em Java, e são interpretadas pela *Dalvik Virtual Machine* e não pela *Java Virtual Machine*. A *Dalvik Virtual Machine* opera transformando Java *byte code* em Dalvik *byte code* em tempo de execução (LECHETA, 2013).

No mesmo nível se localiza a camada *Libraries*, responsável por oferecer suporte a operações com vídeo e áudio, *browser* nativo com suporte a CSS, Javascript, AJAX. Uma bilbioteca que controla todo o armazenamento de dados local, e também suporte a gráficos 3D,

facilitando o desenvolvimento de jogos.

E por fim, a camada *Linux Kernel*, responsável por todo o gerenciamento de memória, *Bluetooth*, *WiFi*, e energia. Como o Android foi desenvolvido com o proprósito de ser executado em *smartphones* com pouca memória e pouca bateria, as bibliotecas de GPU e CPU foram compiladas para código nativo, visando uma melhor performance (BRAHLER, 2010).

Para o desenvolvimento de aplicações na plataforma Android, existem uma série de conhecimentos que um desenvolvedor deve conhecer bem. Essa plataforma oferece diversas classes que facilitam a comunicação entre a aplicação desenvolvida e o sistema operacional. Uma dessas classes é chamada de *Intent*. Uma *Intent* está presente em todo lugar, essa classe representa uma mensagem da aplicação para o sistema operacional. Quando essa mensagem é lançada, cabe ao sistema operacional interpretá-la e oferecer o serviço que foi solicitado (LECHETA, 2013).

Uma *Intent* pode ser utilizada para:

- iniciar um serviço em segundo plano;
- comunicação entre aplicações distintas;
- iniciar uma chamada para um número na agenda de contatos;
- abrir o browser em determinado site;
- e muito mais.

Para capturar os dados de uma *Intent* quando a mesma é lançada, é necessário a configuração de um *BroadcastReceiver*. Um *BroadcastReceiver* é uma classe responsável por reagir a determinados eventos gerados por uma *Intent*. Essa classe é sempre executada em segundo plano, logo uma aplicação pode ser capaz de capturar eventos do sistema operacional sem que o usuário perceba (LECHETA, 2013). Esses eventos podem ser lançados quando o usuário recebe um SMS, quando o mesmo desliga a tela do dispositivo, quando um novo *WiFi* é descoberto, quando a força do sinal do mesmo muda, entre outros.

A classe *BroadcastReceiver* pode ser utilizada também para executar pequenos processos em segundo plano, porém algumas aplicações podem necessitar de um processamento mais demorado, e essa classe não é a mais adequada. O tempo máximo de processamento que um *BroadcastReceiver* pode suportar é de até 10 segundos, passado esse tempo o Android exibirá um erro informando que a aplicação não está mais respondendo.

Para executar processamentos mais demorados a classe mais indicada é a classe *Service*. A classe *Service* é utilizada para executar processamentos longos e com alto consumo de

memória e CPU. Essa classe executa todo o processamento em segundo plano, podendo ou não informar ao usuário sobre o mesmo. Como exemplo de processamento em um serviço, podem se destacar o uso de um player MP3, download de um jogo, imagens ou outras informações. Essa classe é gerenciada pelo sistema operacional e possui prioridade máxima que qualquer processo executando em segundo plano (LECHETA, 2013).

Para utilizar ambos, *Service* e *BroadcastReceiver*, é necesário a configuração do arquivo *AndroidManifest.xml*. Esse arquivo é essêncial para o funcionamento de uma aplicação Android. O mesmo utiliza XML como linguagem de marcação, e é responsável pela configuração de toda a aplicação, onde o programador informa as permissões que a aplicação requer para o seu funcionamento como também o nível mínimo da versão do Android.

A API e a aplicação apresentadas nesse trabalho serão desenvolvidas para essa plataforma, usufruindo de todas as funcionalidades fornecidas pela mesma. Ambas, API e aplicação, irão operar na camada *Applications*.

3.0.3 Tecnicas de Posicionamento Indoor

Essa seção irá apresentar algumas técnicas de posicionamento para sistemas *indoor*, que servirão como base para a API que será desenvolvida nesse trabalho.

3.0.3.1 Proximidade

Utilizando o algoritmo de proximidade, a localização do objeto é simbólica e relativa. Em um primeiro passo, a localização de todas os pontos de acesso de referência são registrados. Quando o dispositivo móvel detecta a presença de uma dessas bases, sua posição é considerada a mesma da base detectada. Quando o dispositivo detecta mais de uma base ao seu redor, sua posição é estimada na base que possui o maior sinal (CONTE et al., 2015).

Esse algoritmo não é de difícil implementação, porém não possui uma precisão muito alta. Para um maior ganho na precisão é necessário que haja muitas estações bases no ambiente (CONTE et al., 2015).

3.0.3.2 RSSI-Based

Essa técninca provê o calculo da distância entre o emissor e o receptor tendo como principal informação a força do sinal recebido (RSSI). A distância pode ser calculada através

da relação entre a potência do sinal e sua taxa de perda em função da distância a qual pode ser obtida pelo cálculo da Perda no Espaço Livre (*Free Space Path Loss*).

Com a equação apresentada abaixo é possível obter a distância entre o receptor e o emissor do sinal.

$$d = 10^{\frac{R-C}{-10n}} \tag{3.1}$$

onde d é a distância estimada, R é a intensidade do sinal medido a 1 metro do emissor ou ponto de acesso. C é o valor da intensidade medido no momento do cálculo. A variável n representa a perda da intensidade do sinal proveniente do emissor. No espaço livre n é igual a 2. Mas para caminhos com obstruções, n tem um valor entre 4 e 5 (CRUZ et al., 2011). A margem de erro da distância obtida varia entre 3 à 30 metros (CONTE et al., 2015).

3.0.3.3 Análise de Cena

A técnica de análise de cena, ou *fingerprinting*, consiste no mapeamento da potência do sinal recebido (RSSI) pelo receptor em locais já conhecidos. Essa técnica é constituída por duas fases: uma *offline* e outra *online*.

A *fase offline* consiste em coletar manualmente o nível do sinal recebido em todo o ambiente a ser mapeado. O projetista percorre todo o ambiente, efetuando a medição do sinal, marcando a posição em um mapa do ambiente (CRUZ et al., 2011).

Na *fase online*, é realizada a leitura da força do sinal recebido e logo depois é feita uma comparação com os dados mapeados na *fase offline*. A posição é estimada como sendo a do sinal com maior similaridade (CRUZ et al., 2011). O cálculo da similaridade pode ser realizado utilizando métodos probabilísticos, técnicas de reconhecimento de padrão, entre outras (LIU et al., 2007).

3.0.4 Padrões de Projeto

3.0.4.1 Singleton

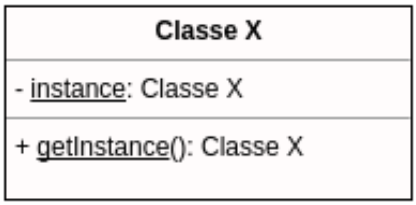
O padrão de projeto *Singleton* garante que em todo o ciclo de vida de uma aplicação, apenas uma intância de determinada classe seja criada. Para algumas aplicações esse padrão de projeto é essêncial, por exemplo, em uma aplicação de bate-papo, deve haver apenas uma classe

que gerência a troca de mensagens entre dois usuários, evitando assim que a lista de mensagens seja duplicada.

Nesse padrão, a própria classe que o implementa, é responsável por manter o controle da única instância. A classe deve garantir que nenhuma outra instância seja criada, como também deve oferecer uma forma de acesso a essa instância (GAMMA, 2009).

A estrutura mais simples de uma classe que implementa esse padrão é a seguinte:

Figura 3 – Estrutura básica do padrão Singleton



Fonte: O próprio autor

Para evitar que essa classe seja instanciada duas vezes, seu contrutor deve ser privado, e o controle de acesso a instância seja fornecido de outra forma. Na Figura 4 é possível observar o atributo instance que é estático. Esse atributo representa a instância única da classe, o mesmo é privado, evitando que outras classes acessem-o diretamente. Apenas usando o método criado pela classe que controla a instância, nesse caso o método getInstance() é possível se obter a instância única da classe.

3.0.4.2 Observer

O padrão *Observer* é utilizado para que quando um objeto mude de estado, todos os seus dependentes sejam informados e atualizados automaticamente (GAMMA, 2009). Esse padrão utiliza basicamente duas classes, uma que será observada e outra que irá observar.

Na linguagem Java, os métodos mais comumente utilizados nesse padrão são:

- registerObserver(Observer observer)
- deleteObserver(Observer observer)
- hasChanged()
- notifyObservers()

O primeiro método é responsável por registrar um *observer* na lista de *observers*. O segundo funciona da forma contrária ao primeiro, removendo um *observer* da lista. O terceiro informa aos *observers* se o objeto observado mudou e por último o método *notifyObservers*

notifica todas os *observers* quando há uma mudança no objeto que está sendo observado.

A vantagem de uso desse padrão de projeto é que tanto as classes que observam quanto as que são observadas podem ser reutilizadas. Ambas implementam interfaces, viabilizando a reutilização. A desvantagem fica evidente quando há muitas mudanças e muitos observadores, logo a performance da aplicação pode ser afetada, pois pode haver uma grande quantidade de mensagens entre essas classes.

4 PREDETECT

Nessa seção será apresentada toda a arquitetura da API desenvolvida explicando todo o seu fluxo de funcionamento.

4.0.1 Detector de Presença (PreDetect)

Antes de todo desenvolvimento fez-se necessário o levantamento de todas as funções que seriam disponibilizadas pela API e sua arquitetura. Para verificar a viabilidade da contrução da mesma, foi necessário obter um conhecimento mais profundo sobre as funcionalidades que a plataforma Android oferece, desde informações relacionadas a coleta de informações das redes *WiFi*, até serviços que funcionam em sengundo plano.

4.0.1.1 Arquitetura da API

Para a construção da aplicação foi usada a API de *Network* da plataforma Android. A partir dessa API, se torna possível obter dados das redes *WiFi* próximas a determinado dispositivo. Para ter acesso a esses dados é necessário informar ao Android que a aplicação está interessada nos mesmos, logo, é necessário a criação de um *BroadcastReceiver* que se trata de uma classe onde é possível obter dados vindos de eventos gerados pelo sistema ou por outras aplicações. Um exemplo de evento vindo do sistema ocorre quando o usuário desliga ou liga a tela do dispositivo, ou quando uma nova rede *WiFi* é descoberta pelo mesmo. Esses eventos são disparados pelo sistema operacional, e todos os *broadcasts* que estiverem configurados para receber esse evento, são notificados.

Para que a API tenha acesso as informações dos roteadores *WiFi*, a classe *NetworkReceiver* foi implementada. Antes de codificar essa classe foi necessário alterar as configurações no arquivo *AndroidManifest*.

A configuração da classe *NetworkReceiver* no arquivo *AndroidManifest* possui a seguinte estrutura:

```
1 <receiver android:name="danielfilho.ufc.br.com.predetect.
    receivers.NetworkReceiver" android:enabled="true"
    android:exported="true">
2     <intent-filter android:priority="1000">
3         <action android:name="android.net.wifi.RSSI_CHANGED
```

```
            " />
        <category android:name="android.intent.category.
            DEFAULT"/>
    </intent-filter>
    <intent-filter>
        <action android:name="danielfilho.ufc.br.com.
            predetect.OBSERVING_ENDS"/>
        <category android:name="android.intent.category.
            DEFAULT"/>
    </intent-filter>
</receiver>
```

Nessa configuração é informado que a classe *NetworkReceiver* é um *receiver*, ou seja, ela é uma classe que herda de *BroadcastReceiver*. Além disso, outras informações foram adicionadas, como os eventos que essa classe tem interesse de ser informada, sendo eles, RSSI_CHANGED, que é um evento gerado pelo sistema, quando o indicador de potência do sinal (*RSSI*) de algum *WiFi* próximo ao dispositivo móvel muda. Com esse evento se torna possível observar a mudança da potência do sinal de cada aparelho *WiFi* próximo ao dispositivo. Outro evento que essa classe está configurada para receber é o evento OBSERVING_ENDS, responsável por informar que o serviço de monitoração de presença terminou, esse evento será abordado nas próximas seções.

A classe *NetworkReceiver* foi configurada para receber os dados quando a força do sinal do *WiFi* muda. Quando esse evento é lançado, a classe *NetworkManager* inicia seu trabalho. Quando o indicador de força de sinal de algum *WiFi* muda, a classe *NetworkReceiver* é notificada pelo sistema operacional e repassa essa notificação para a classe *NetworkManager*. Quando essa última é notificada, o processo de obtenção dos dados das redes *WiFi* se inicia. Para se obter esses dados se utiliza a API *WifiManager*, fornecida pela a plataforma Android. Com essa classe é possível obter o *MAC*, *SSID*, *RSSI*, entre outros dados das redes *WiFi* que não são relevantes para a *API* proposta.

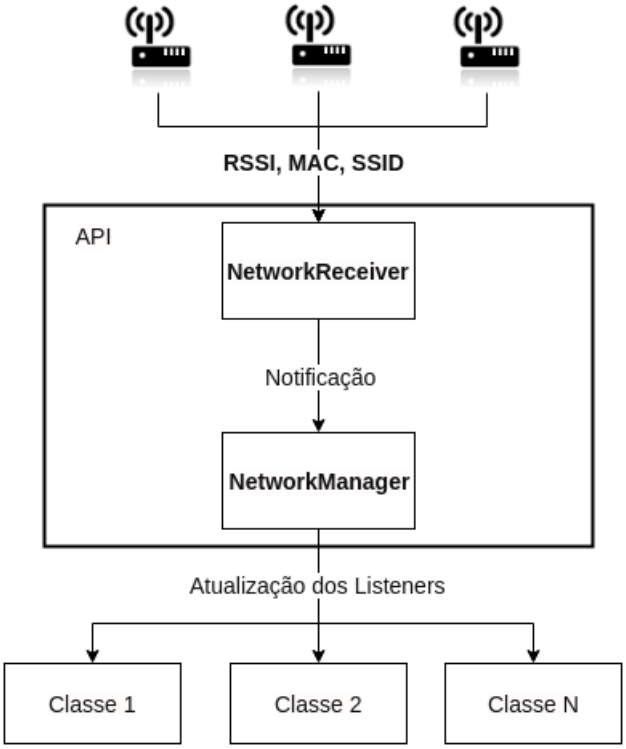
Todos os dados são repassados para a classe *NetworkManager*, onde os mesmos são encapsulados em uma lista de objetos pertencentes a classe *WiFiData*, essa última, possui os atributos RSSI, MAC, SSID e distance. A distância é calculada pela classe *NetworkManager* no

momento da criação desses objetos. Essa conversão é provida pela transformação do *RSSI* em distância utilizando a fórmula 3.1.

A classe *NetworkManager* implementa o padrão de projeto *Singleton* possuindo assim, apenas uma instância em todo o tempo de vida da aplicação. Outro padrão de projeto que essa classe implementa é o *Observer*, e este último foi implementado para que uma aplicação que utiliza a API tenha acesso a lista de dados do *WiFi* próximos, inclusive a sua distância aproximada entre o aparelho e o *smartphone*. A classe interessada nesses dados deve obter uma instância de *NetworkManager*, implementar a interface *NetworkListener* e registrar seu interesse em observar o comportamento das redes *WiFi* utilizando o método *registerListener* da instância obtida. Quando um evento é repassado para *NetworkManager*, todos as outras classes que estão observando seu comportamento são informadas da mudança.

Todo o processo exemplificado anteriormente pode ser visualizado na Figura 4

Figura 4 – Disponibilização dos dados das redes *WiFi*



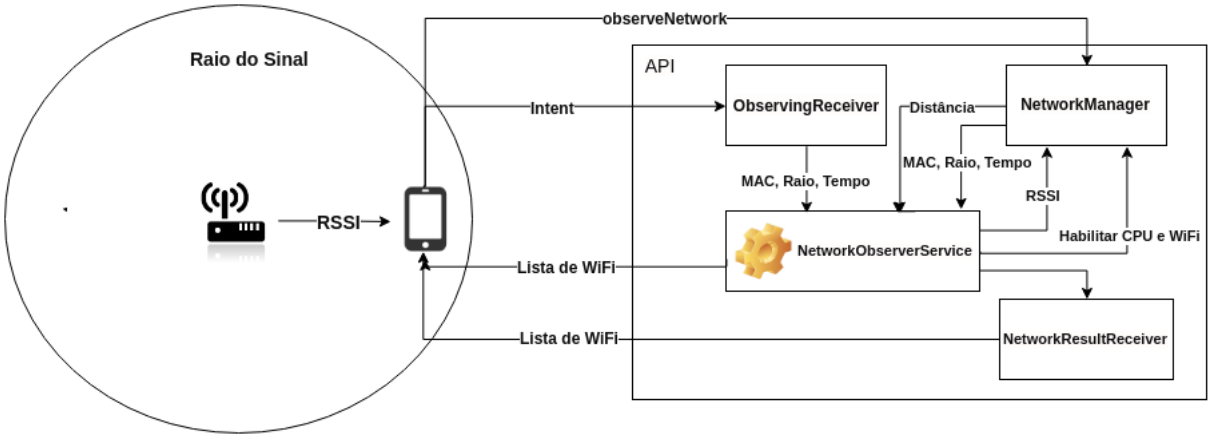
Fonte: O próprio autor

Outra funcionalidade fornecida pela API é de checar se um dispositivo está presente dentro de um raio de distância em um tempo especificado. Com essa funcionalidade se torna possível saber a porcentagem de tempo que um dispositivo ficou próximo de determinada rede *WiFi*. Toda a sua implementação utiliza a classe *Service* fornecida pela a plataforma Android,

logo a sua execução será em segundo plano.

A Figura 5 apresenta o fluxo de funcionamento dessa classe.

Figura 5 – Detecção de Presença



Fonte: O próprio autor

Existem duas formas de requisitar esse serviço. Para primeira forma é necessário implementar a interface *WiFiObserver* e sobrescrever o método *onWiFiObservingEnds(final int resultCode, final List<WiFiData> list)*, que como argumento, se tem um código de resultado informando se o serviço foi executado com sucesso ou não, e uma lista de *WiFiData* onde estão todos os dados , incuindo a porcentagem de tempo que o *smartphone* ficou próximo dos *WiFi* usados como referência. Feito isso, uma instância da classe *NetworkManager* deve ser obtida sendo possível executar o método *observeNetwork(WiFiObserver observer, List<WiFiData> wiFiDatas, int time, double rangeDistance)* passando a instância da classe que implementa *WiFiObserver*, uma lista de *WiFiData* em que, para cada objeto dessa classe, é preciso informar o *MAC* do *WiFi* que será utilizado como referência, em seguida, um inteiro informando o tempo em milisegundos, e um raio de distância mínimo entre o *smartphone* e o dispositivo *WiFi*.

A segunda forma para a requisitar esse serviço é lançando uma *Intent* com os dados do raio distância do *WiFi* para o *smartphone*, como também o tempo de checagem e por último uma lista de *MACs* de referência. Quando essa *Intent* é lançada o *BroadcastReceiver ObservingReceiver* é notificado. Este último inicia o serviço *NetworkObservingService* em segundo plano e repassa todos os dados vindos da *Intent*. Diferente da outra forma de requisição do serviço, nessa forma quem o inicia é a classe *ObservingReceiver* e não *NetworkManager*.

A classe *NetworkObservingService* é simplemente uma *thread* que, a partir de um tempo configurado, captura os *WiFi* proximos, verifica se o *MAC* informado está entre os *WiFi*

capturados, logo depois calcula a distância para cada um deles e verifica se o dispositivo está dentro da distância mínima informada. No final da verificação é retornado a porcentagem de tempo que o dispositivo ficou presente dentro da distância especificada.

Para evitar o consumo de energia, quando um dispositivo não está em uso, a plataforma Android possui um mecanismo de gerenciamento de bateria que desabilita todos os seus componentes, inclusive a CPU (KIM; CHA, 2013). Porém esse mecanismo deve ser utilizado com cuidado, caso alguma aplicação esteja fazendo donwload de algum dado da internet, seu comportamento será afetado. Para evitar que durante o serviço de detecção de presença o dispositivo utilizado desabilite a CPU foi necessário o gerenciamento de um *WakeLock*. *WakeLock* é um mecanismo que garante que o dispositivo ficará ativo mesmo depois de muito tempo de inatividade quando a aplicação está executando.

Existem diferentes tipos de *WakeLock* a Tabela 1 apresenta a lista de opções disponíveis.

Tabela 1 – Tipos de WakeLock

Constante	CPU	Tela	Teclado
PARTIAL_WAKE_LOCK	Ligado	Desligada	Desligado
SCREEN_DIM_WAKE_LOCK	Ligado	Parcialmente	Desligado
SCREEN_BRIGHT_WAKE_LOCK	Ligado	Ligada	Desligado
FULL_WAKE_LOCK	Ligado	Ligada	Ligado

O desenvolvedor tem a liberdade de informar qual o tipo de *WakeLock* que será usado, utilizando as constantes apresentadas. A constante PARTIAL_WAKE_LOCK informa que apenas a CPU será habilitada tendo a tela e o teclado desabilitado. A segunda constante, SCREEN_DIM_WAKE_LOCK, informa que a CPU estará ativa e a tela do dispositivo estará parcialmente ligada. Já a quarta constante, SCREEN_BRIGHT_WAKE_LOCK, apenas o teclado estará desabilitado e a tela do dispotivo e a sua CPU estarão em funcionamento. Na última constante, FULL_WAKE_LOCK, todos os componentes do dispositivo estarão em completo funcionamento.

Quando requisitado, o serviço de detecção de presença, implementado na API utilizará apenas o *WakeLock* que ativa somente a CPU junto com o *WiFiWakeLock*, que funciona de forma silimar ao *WakeLock*. Logo, mesmo se o dispositivo ficar sem a interação do usuário, a CPU e o *WiFi* não serão desabilitados.

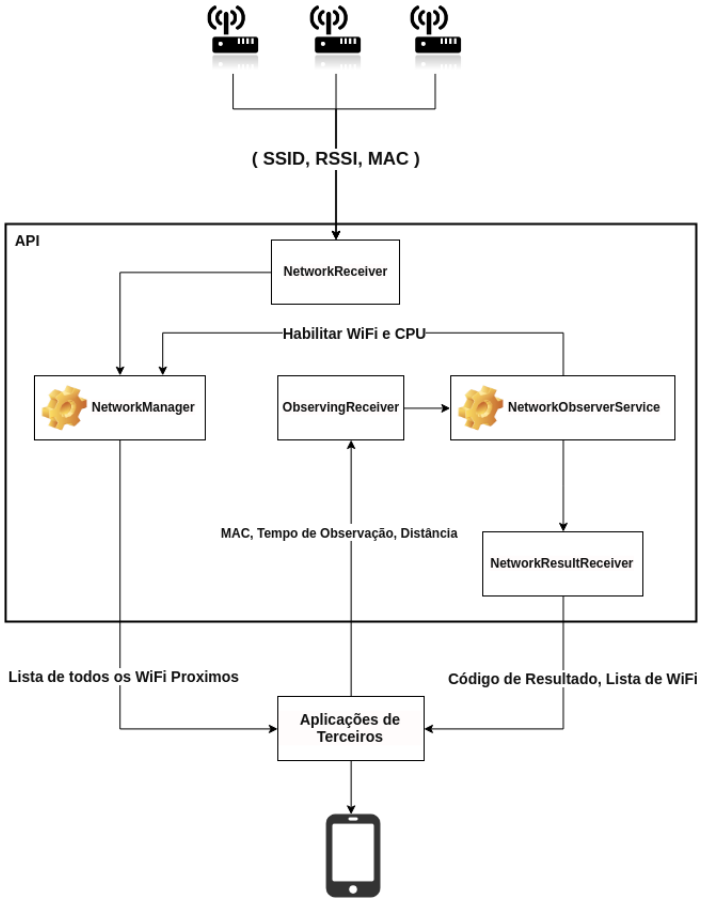
A classe responsável por todo o gerenciamento dos *WakeLock* ativos, é a *NetworkManager*. Primeiramente, quando o serviço de detecção de presença está para iniciar

seu processamento, é requisitado a criação dos *WakeLock* para a CPU e *WiFi*. Para evitar o consumo indevido de bateria, quando o serviço é finalizado, ambos os *WakeLock* são liberados.

Um ponto a se destacar é que não é necessário o *smartphone* estar conectado a rede, já que esses dados de *RSSI*, *MAC* e *SSID* são disponibilizados sem haver a conexão entre os mesmos.

Uma visão geral da arquitetura da API com todas as funcionalidades descritas anteriormente, pode ser visualizada na Figura 6.

Figura 6 – Visão geral da API



Fonte: O próprio autor

4.0.1.2 Documentação da API

Para que outros desenvolvedores tenham acesso e utilizem a API, foi necessário disponibiliza-lá em um repositório público. A plataforma escolhida para disponibilização de todo código e de sua documentação foi a plataforma *GitHub*.

Todo código e sua documentação pode ser acessado em <https://github.com/dielfilho/PreDetect>. Com o código disponível na internet fica possível que outros desenvolvedores

colaborem com o projeto.

4.0.2 Desenvolvimento da Aplicação de Presença

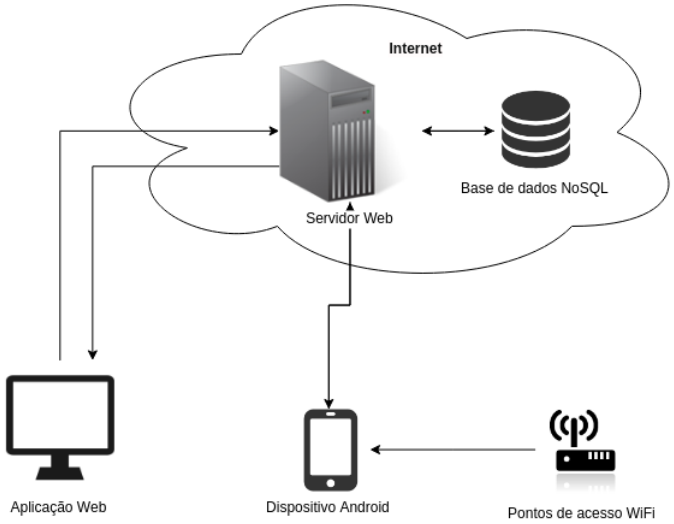
Com essas duas funcionalidades da API ficou possível desenvolver a aplicação para detecção de presença. O desenvolvimento dessa aplicação foi dividido em duas etapas. A primeira etapa consiste no desenvolvimento de uma aplicação cliente para a plataforma Android e outra para a plataforma *WEB*, ambas aplicações serão abordadas nessa seção.

A arquitetura do sistema será constituída pelos seguintes elementos base:

- Servidor Web com acesso a base de dados NoSQL
- Aplicação móvel para dispositivos Android
- Aplicação Web
- Pontos de acesso *WiFi*

Na Figura 7 é possível observar como os elementos se comunicam

Figura 7 – Comunicação entre as partes da aplicação



Fonte: O próprio autor

Com base na Figura 7 pode-se afirmar que aplicação está dividida em quatro partes: i) Aplicação móvel, onde o usuário terá acesso a todas as suas turmas, e onde a sua presença será computada. ii) Aplicação Web onde é possível cadastrar turmas e horários de checagem de presença. iii) Servidor Web onde todas as requisições vindas das outras aplicações serão tratadas e salvas em uma base de dados NoSQL. iii) Pontos de acesso *WiFi* que serão responsáveis por servirem como pontos de referência na checagem de presença da aplicação cliente.

4.0.2.1 Aplicação Servidor

O servidor Web tem por objetivo tratar todas as requisições vindas das outras aplicações permitindo o acesso aos dados salvos na base de dados. Em termos de tecnologias utilizadas no desenvolvimento, optou-se pelo o uso da plataforma NodeJs por ser uma plataforma gratuita, fácil de se manipular e robusta.

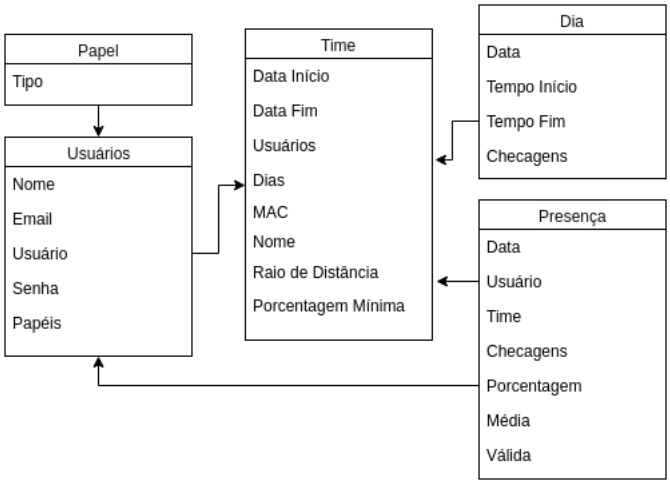
NodeJs é uma plataforma desenvolvida sobre o motor Javascript do Google Chrome (PEREIRA, 2014). Com essa plataforma é possível adicionar módulos que facilitam o desenvolvimento de aplicações escaláveis, um dos módulos utilizados foi o *Express* que facilita o gerenciamento de requisições HTTP com criações de rotas, cada requisição feita, é direcionada para uma rota específica.

Para o armazenamento dos dados foi utilizado o banco de dados MongoDB. MongoDB é um banco de dados NoSQL, orientado a documentos, seus dados, diferentemente dos bancos de dados relacionais, não são salvos em tabelas com linhas e colunas, e sim em documentos. Um documento é basicamente um objeto JSON com objetos e listas internas associados ao mesmo (WEI-PING; MING-XIN; HUAN, 2011).

No NodeJs a comunicação com o MongoDB se torna possível através de um textitdriver específico. Essa comunicação pode ser facilitada com o uso de um ORM chamado Mongoose. Com o Mongoose o acesso a base de dados se torna mais fácil.

Utilizando o Mongoose, primeiramente é necessário fazer todo o mapeamento das entidades envolvidas. O mapeamento das entidades pode ser vizualizado na Figura 8

Figura 8 – Entidades do banco de dados



Fonte: O próprio autor

Nessa figura é possível observar todas as entidades modeladas para a aplicação de presença. Cada uma dessas entidades são representadas em forma de documentos no banco de dados, a entidade Usuário possui os dados básicos, como nome email, usuário e senha, além desses atributos, um usuário também possui um papel, que delimita as ações que o mesmo pode efetuar nas aplicações. Esse papel é dividido em Administrador e Estagiário. Um administrador é responsável pelo gerenciamento das turmas e dos horários de checagem de presença. Já um estágiário pode somente, olhar os dados das turmas que o mesmo está vinculado, além de visualizar sua presença em seu dispositivo.

Um estagiário pode ser vinculado a um time, onde o mesmo possui atributos como data de início e fim, que representa o seu período de atividades, como também uma lista de estagiários que o compõem. Além do período de atividades, é necessário informar os dias de trabalho desse time. A entidade Dia possui os horários de início e fim de um dia específico de trabalho, como também os horários e durações de checagem de presença, que são utilizados pela a aplicação móvel. Além dos dias, um time possui um *MAC* de um ponto de acesso *WiFi* que será utilizado como referência para checagem de presença na aplicação móvel. Cada time possui um raio em que os dispositivos móveis poderão ficar para que suas presenças sejam confirmadas, além de uma média de porcentagem que informará se o estagiário está presente ou não.

A entidade Presença é utilizada para registrar as presenças de cada estagiário, essa entidade possui atributos como data da presença, o usuário (estagiário) e o time vinculado a ela, além da quantidade de checagens que são feitas no dia, a porcentagem para cada checagem, seguidas da média dessas checagens e por fim a validade dessa presença, que informa se a média da presença é maior ou igual que a média de porcentagem cadastrada no time.

Todas essas entidades foram modeladas utilizando a ferramenta Mongoose e são utilizadas tanto na aplicação móvel quanto na Web.

Quando uma requisição é feita, seja pela aplicação móvel ou pela Web, o servidor deve ser capaz de tratá-la. Esse tratamento é feito utilzando o roteamento provido pelo módulo *Express*, onde se informa o tipo de requisição HTTP e a função que irá tratar essa requisição. Várias rotas foram criadas, para ter acesso aos dados salvos na base de dados.

4.0.2.2 Aplicação Web

Uma aplicação Web foi desenvolvida para para oferecer uma interface amigável para o acesso dos dados no servidor. Essa aplicação foi desenvolvida utilizando o *framework*

AngularJs junto com CSS e HTML para a estruturação das páginas. AngularJs é um *framework* de desenvolvimento de aplicações Web sob o padrão MVC (*Model-View-Controller*) utilizando a linguagem Javascript, sendo uma plataforma de código aberto mantida pelo Google. Utilizando esse *framework* é possível criar componentes reutilizáveis, fazer integração com servidor, escrever testes automatizados, entre outras funcionalidades.

Essa aplicação possui os seguintes elementos:

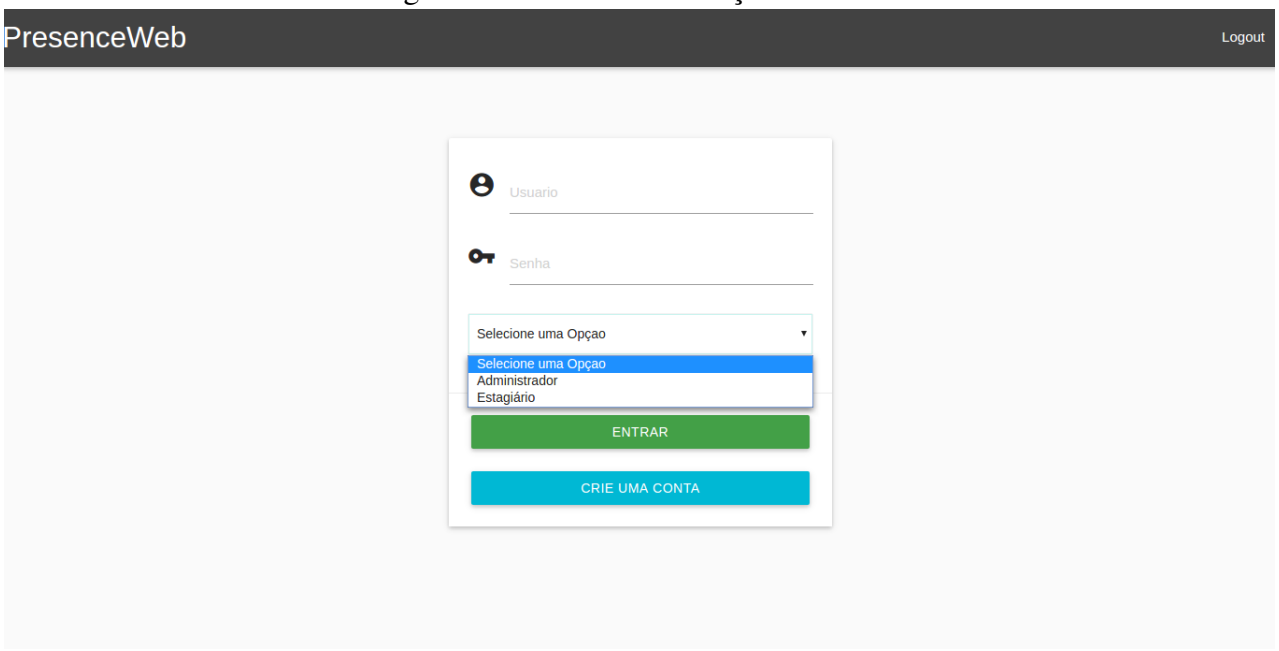
- Controladores (*Controllers*)
- Serviços (*Services*)
- Rotas

Os controladores possuem o papel de tratar tudo que é apresentado na camada da visão. Os Serviços são responsáveis por fazer a comunicação com os servidor e as Rotas possuem a responsabilidade de direcionar o usuário para o local correto com base na URL informada.

Três controladores foram criados, um para a página de autenticação, um para a página do administrador e outro para a página do estagiário. O controlador da página de autenticação é responsável por requisitar todos os papeis salvos no servidor e mostrá-los para o usuário na hora do *login*. Quando um usuário tenta acessar o sistema informando os seus dados de usuário e senha, esse controlador tem acesso a esses dados e encapsula em um objeto json e envia para o servidor utilizando o serviço de *login*, caso o usuário informe as credenciais corretas o controlador acessa as rotas e o direciona para a página correta com base no seu papel.

A Figura 9 apresenta a tela de autenticação da aplicação *Web*.

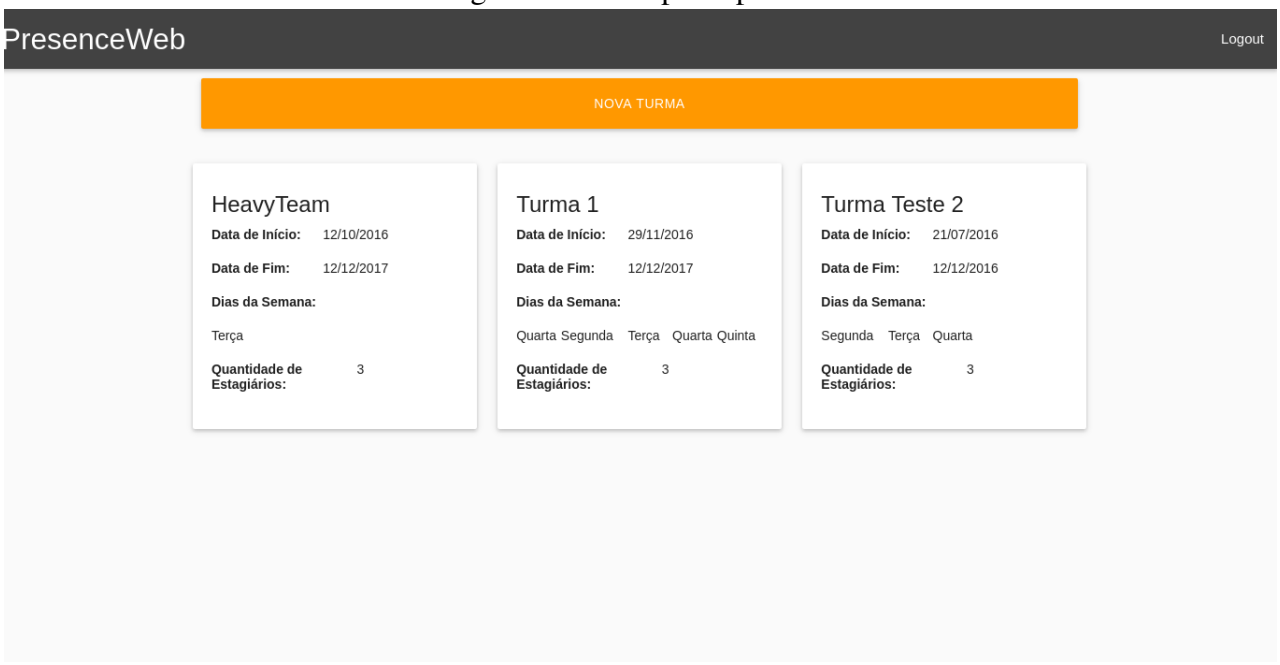
Figura 9 – Tela de autenticação Web



Nessa tela também é possível ter acesso a tela de criar uma conta. Nessa tela são informados os dados de email, nome completo usuário e senha de acesso. Vale lembrar que nessa tela o cadastro é apenas destinado aos estagiários.

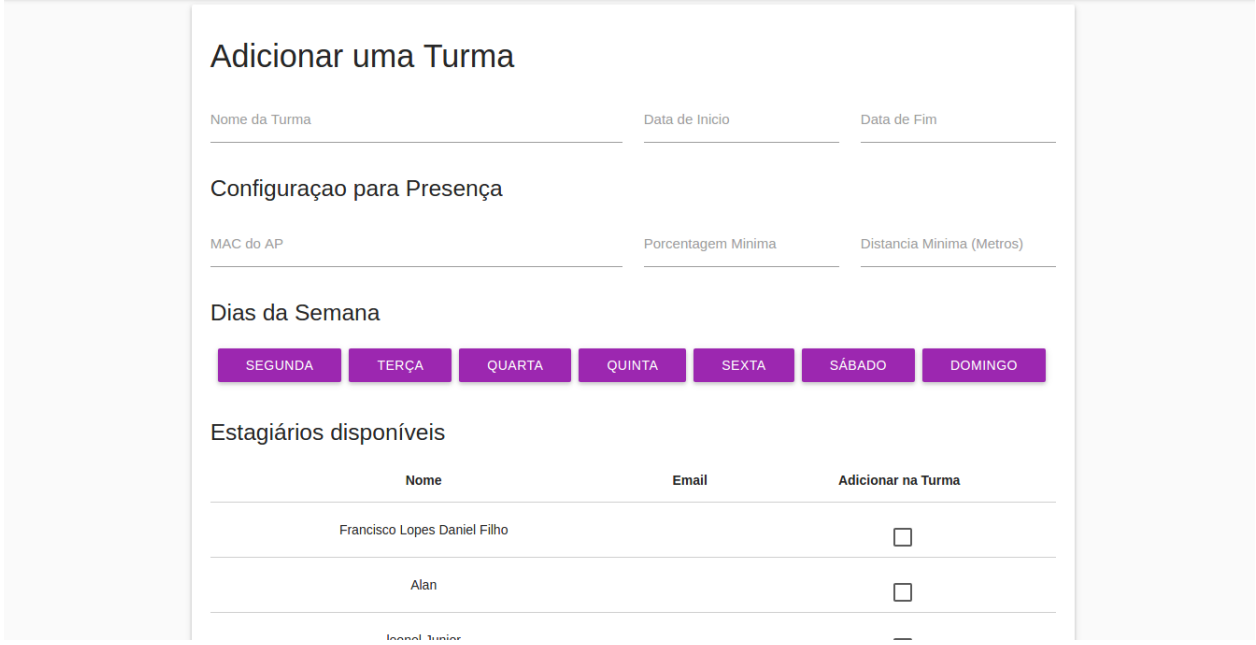
Quando um administrador se autentica no sistema, ele é levado para a sua tela e o controlador é alterado para o *AdminController*. Esse controlador é responsável por administrar todas as ações que o administrador pode fazer, como cadatrar turmas e visualizar as cadastradas. Na Figura 10 apresenta a tela inicial do papel de administrador

Figura 10 – Tela principal



Nessa tela é possível visualizar todas as turmas cadastradas com os dados de data de início e fim das atividades, dias da semana, e a quantidade de estagiários de cada turma. Quando o administrador clica no botão Nova Turma, o mesmo é direcionado para tela de cadastro de turmas.

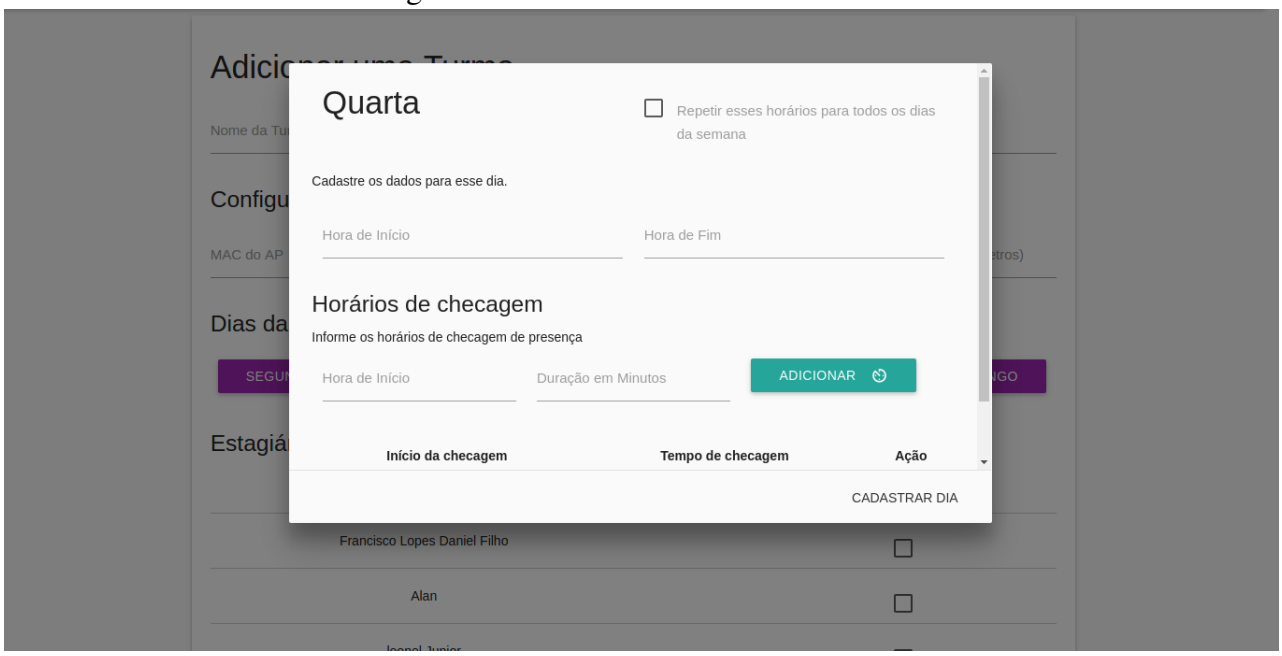
Figura 11 – Tela de cadastro de turmas



Na tela de cadastro de turmas, apresentada na Figura 11, o administrador irá informar os dados da turma como nome, data de início e fim de atividades e logo depois os dados relacionados a configuração do ponto de acesso *WiFi*, onde é cadastrado o *MAC* do aparelho que será tido como referência. Depois de informar o *MAC*, o administrador deve informar a porcentagem mínima de presença que informará se a mesma é valida ou não.

Além dos dados informados anteriormente o administrador deve cadastrar os horários de atividade do time para cada dia da semana. Quando o administrador clica em um dia, um modal é aberto. Esse modal pode ser visualizado na Figura 12.

Figura 12 – Tela de cadastro de dias



Nessa tela é possível cadastrar os horários de início e de fim de atividades para o dia escolhido. Logo depois é necessário informar os horários de checagem de presença, indicando a hora de inicio da checagem e a duração dessa checagem. Essas checagens serão utilizadas na aplicação móvel para determinar se o estagiário está presente ou não.

4.0.2.3 Aplicação Móvel

Essa seção irá apresentar a aplicação desenvolvida para a plataforma Android utilizando a API proposta. Serão apresentadas as telas principais bem como o modo de funcionamento e todas as dificuldades encontradas em todo o seu desenvolvimento.

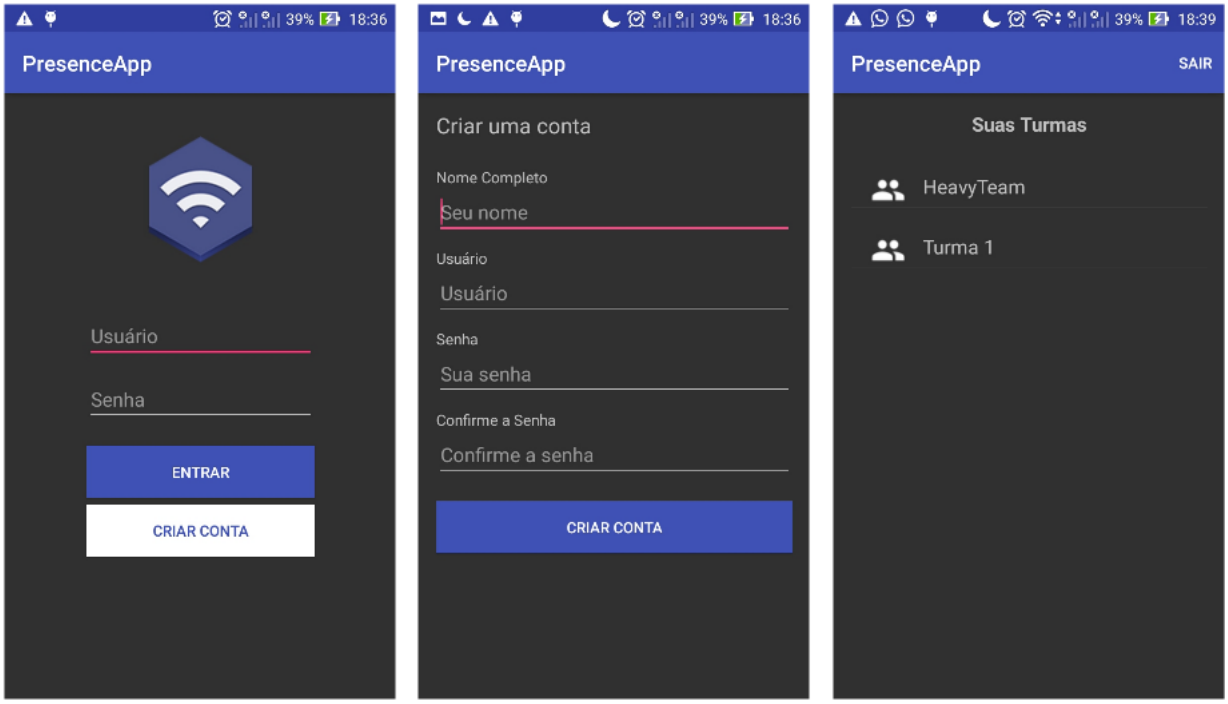
Toda a codificação do projeto foi feita utilizando a *IDE* AndroidStudio pois a mesma

oferece todo o suporte necessário para o desenvolvimento de aplicações Android.

4.0.2.3.1 Telas da Aplicação Móvel

Antes de abordar o funcionamento interno da aplicação, as telas principais serão apresentadas.

Figura 13 – Telas da Aplicação Móvel



A primeira tela representa a tela de autenticação, é nessa tela onde o estagiário informa suas credenciais, se o mesmo ainda não possuir um conta, é possível o cadastro clicando no botão "Criar Conta", que irá levá-lo para a segunda tela. Na segunda tela o estagiário poderá informar todos os seus dados. Quando sua conta é criada, o mesmo é direcionado para tela inicial onde todas as suas turmas são apresentadas.

4.0.2.4 Arquitetura da Aplicação

Quando um usuário acessa a tela inicial da aplicação, uma requisisão por todas as turmas é feita para o servidor Web. A comunicação entre essas duas aplicações e feita utilizando objetos JSON. Quando o servidor responde, todo o JSON é convertido em objetos da linguagem Java utilizando a biblioteca GSON, que faz todo o processamento de conversão.

Um padrão para as repostas do servidor foi elaborado facilitar a comunicação entre as partes.

```
1 {result:true, data:{}]
```

A propriedade *result* pode assumir dois valores: *true* ou *false*, esses valores informam se o processamento feito no servidor foi executado com sucesso. A propriedade *data* representa um objeto ou uma lista de objetos retornados pelo servidor.

Quando o servidor retorna a lista de turmas, é necessário agendar os horários de checagem de presença cadastrados pelo administrador. Esse agendamento é feito utilizando uma classe disponibilizada pela plataforma Android. Essa classe é chamada *AlarmManager*. Com essa classe é possível agendar a execução de um processamento em determinado momento no futuro. Para isso uma *Intent* é utilizada para enviar uma mensagem ao sistema em uma data e hora desejada (LECHETA, 2013).

A vantagem de utilizar um alarme é que depois que ele é ativado, mesmo se o celular ficar inativo, o alarme continuará ativo, pronto para disparar na data e hora especifica. Somente quando o dispositivo for reiniciado, que todos os alarmes cadastrados serão cancelados (LECHETA, 2013). O nome dessa classe pode gerar confunsão em relação a sua funcionalidade, ela é reponsável por agendar processamentos e não por agendar o alarmes sonoros no dispositivo.

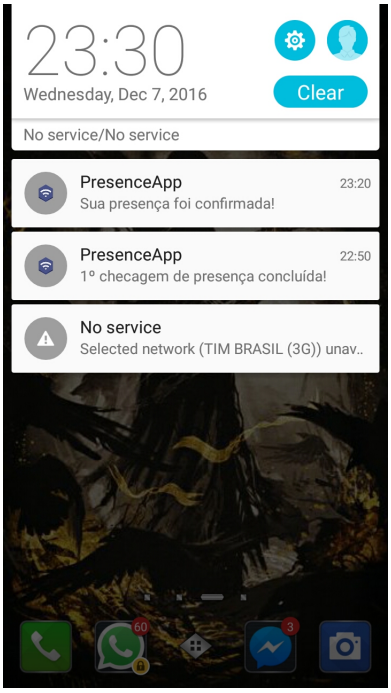
Para cada turma retornada pelo servidor é necessário se obter todos os seus horários de checagem de presença junto com o *MAC* do ponto *WiFi* e encapsular todos em uma *Intent*, que será disparada futuramente. Quando essa *Intent* é disparada pelo alarme, o serviço de observação de presença da API é executado.

No fim da execução do serviço da API, outra *Intent* é lançada no sistema operacional, informando a porcentagem de presença do dispositivo móvel em relação ao ponto de *WiFi*. A aplicação de presença está configurada para receber essa *Intent*. Logo essa porcentagem é adquirida pela aplicação de presença e enviada para o servidor. Quando essa porcentagem é enviada para o servidor, primeiramente, é feita uma verificação se a checagem de presença recebida é a última do dia, se for, uma média de presença é calculada e a partir dessa média é possível saber se a presença do estagiário é válida ou não. Caso não seja a última checagem do dia, o servidor responde informando que a checagem foi salva e está pronto para receber a próxima.

Para confirmar sua validade outra comparação é feita obtendo a média de todas as

checagens e verificando se essa média é maior ou igual a porcentagem mínima cadastrada pelo administrador. Ao final dessa computação, uma resposta é retornada para a aplicação, onde uma notificação é apresentada ao usuário. A Figura 14 apresenta as notificações de presença.

Figura 14 – Notificações de Checagem



5 RESULTADOS

Essa seção irá apresentar todos os resultados obtidos com o desenvolvimento e teste da API e da aplicação de detecção de presença.

5.0.1 Teste da API

Utilizando a API foi possível verificar o comportamento das ondas de um ponto de acesso *WiFi*, assim como a precisão do cálculo da distância para o mesmo. Um teste foi elaborado para verificar a variação do indicador de força do sinal recebido (*RSSI*) e consequentemente a variação do cálculo da distância.

O *smartphone*, com uma aplicação de teste que utiliza a API desenvolvida, foi colocado em uma posição fixa dento de uma sala, à quatro metros de distância do ponto de acesso *WiFi*. Durante a coleta nenhuma barreira impediu a passagem do sinal. O mesmo ficou coletando os dados de *RSSI* e efetuando o cálculo da distância durante trinta minutos.

Todos os dados coletados foram salvos pela aplicação de teste em um arquivo no formato CSV. No final da coleta, todos essas dados foram importados para o PostgreSQL, onde consultas foram feitas para o esbouço dos gráficos. Uma amostra dos dados coletados pode ser visualizada na Figura 15.

Figura 15 – Estrutura dos dados da coleta

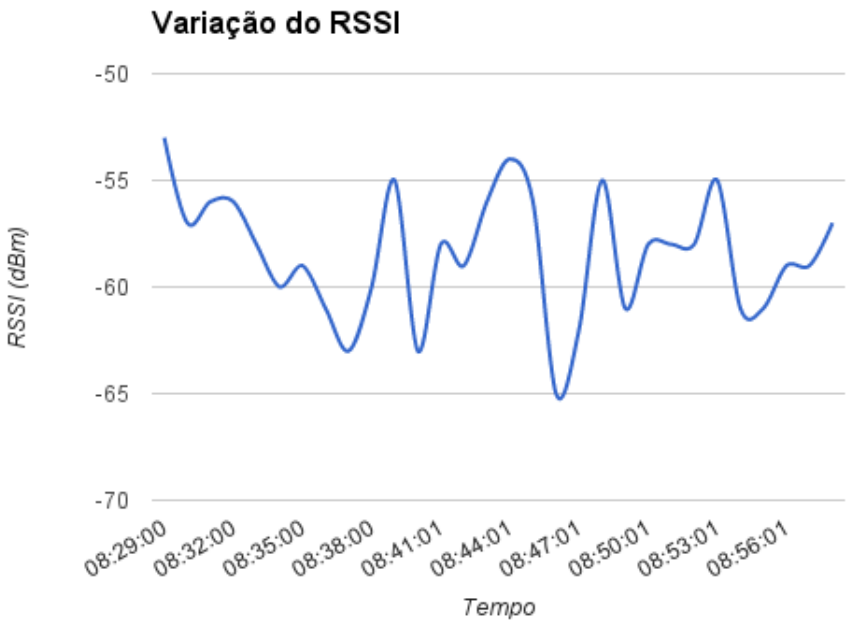
	timelog bigint	mac character varying(255)	ssid character varying(255)	rssi integer	distance numeric
1	1481196540812	34:bf:90:ce:85:b0	brisa-100870	-53	2.7
2	1481196540813	d0:04:92:03:c7:98	brisa-118992	-75	11.4
3	1481196540814	d0:04:92:03:c4:d0	brisa-118769	-74	10.7
4	1481196540815	d0:04:92:07:d1:68	brisa-125851	-75	11.4
5	1481196540816	34:bf:90:ce:c9:e0	CEDRO	-81	16.9
6	1481196600821	34:bf:90:ce:85:b0	brisa-100870	-57	3.5

Fonte: O próprio autor

Nessa tabela é possível armazenar a hora da coleta, o *MAC* do aparelho *WiFi*, o nome da rede (*SSID*) o indicador de força do sinal recebido (*RSSI*) e a distância calculada pela API no momento da coleta. A cada minuto a API coletou dados de todos os *WiFi* próximos.

O resultado da coleta pode ser visualizado no gráfico a seguir:

Figura 16 – Gráfico da Variação do RSSI

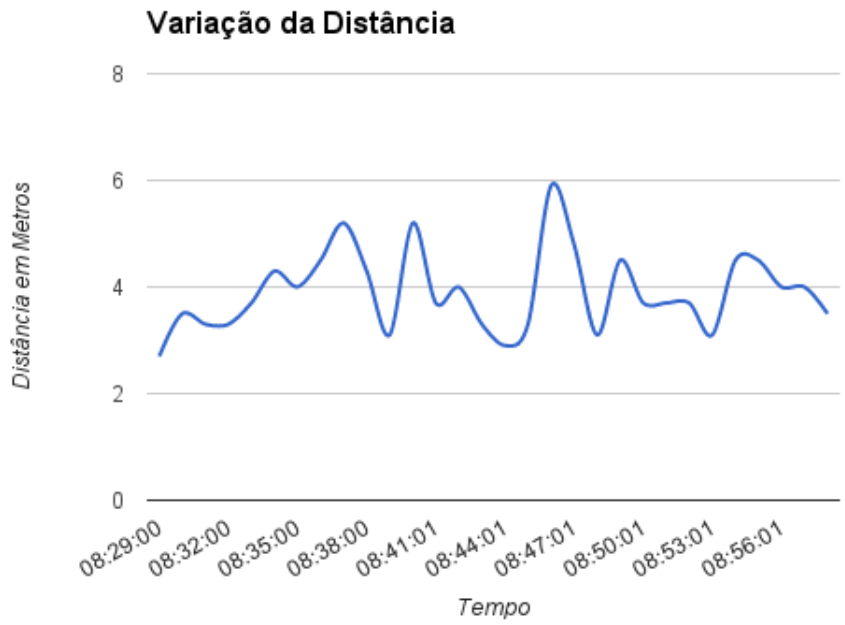


Fonte: O próprio autor

É possível notar no gráfico apresentado que o *RSSI* varia bastante mesmo o *smartphone* estando em uma posição fixa, além da ausência de barreiras entre ele o ponto de acesso. Essa variação afeta diretamente na precisão da distância fornecida pela API desenvolvida. Ao total foram coletados dados de dezoito pontos de acesso *WiFi*. Porém esse gráfico apresenta somente a variação do ponto de acesso mais próximo.

Durante a coleta do *RSSI* a API fez a conversão do mesmo em distância. O gráfico a seguir apresenta a variação da distância durante trinta minutos de coleta.

Figura 17 – Gráfico da Variação da distância



Fonte: O próprio autor

Neste gráfico é possível observar que a distância entre o *smartphone* e o ponto de acesso *WiFi* varia bastante. No teste o *smartphone* ficou posicionado à uma distância de quatro metros, mesmo estando parado, o cálculo da distãncia variou consideravelmente. A média da distância coletada foi de 3,91 metros a qual é bastante próxima da esperada. Mesmo com essa média bastante próxima da correta, deve-se levar em consideração os picos de mudança do sinal, pois os mesmos afetam aplicações que requerem uma acurácia maior.

5.0.2 Teste da Aplicação de Presença

Para verificar se a aplicação de presença funciona de forma esperada, foram feitos testes em duas salas, essas salas estão localizadas na UFC - Campus Quixadá e nelas estão alocados projetos onde os estudantes do campus podem vivenciar uma experiência de estágio.

Uma conta de administrador foi criada e a partir dela foi possível cadastrar novas turmas para teste. Se utilizando de um ambiente de estágio em uma sala no campus, foi pedido para que três participantes instalassem o aplicativo e criassem uma conta a partir do mesmo. Logo depois um ponto de acesso *WiFi* foi posicionado dentro da sala. O *MAC* desse ponto foi registrado, junto com os dias de atividade na semana, como também os horários de checagem de presença para cada dia, um ráio de distância de cinco metros, e uma porcentagem mínima para

validar a presença.

Quando os participantes entraram no aplicativo todos os dados da turma foram requisitados para o servidor Web. Com esses dados todas os horários de checagens de presença foram agendados e podem ser visualizados na Tabela 2.

Tabela 2 – Checagem de Presença Sala de Projetos

Participante / Horários de Checagem	**14:40**	**15:20**	**15:40**	**Média Final**
Participante 1	100%	100%	100%	100%
Participante 2	99%	100%	94%	97,6%
Participante 3	94%	68%	100%	87,3%

A Tabela 2 apresenta checagens de presença de um dia com os três participantes selecionados. Três horários de checagem foram cadastrados, cada um com um tempo de duração de vinte minutos. No final da checagem, é calculado a média das checagens para cada participante. Todos os participantes estiveram presentes durante todo o tempo de checagem.

Outro teste foi executado no Núcleo de Práticas de Informática (NPI), que funciona dentro do campus da UFC - Quixadá. Neste teste teve a colaboração de dois participantes, ambos com dispositivos diferentes. O resultado das checagems pode ser visualizado na Tabela 3

Tabela 3 – Checagem de Presença NPI

Participante / Horários de Checagem	**13:40**	**16:00**	**17:00**	**Média Final**
Participante 4	100%	100%	99%	99,6%
Participante 5	0%	0%	0%	0%

Nesse teste, ambos os participantes estavam presentes em todas as checagens, porém, como pode ser visto na Tabela 3, mesmo presente, o Participante 5 não obteve sua presença. Isso pode ter vindo a acontecer por vários fatores, dentre eles, o dispositivo não está conectado em nenhum ponto de acesso a internet, como as presenças não são salvas localmente no *smarphone*, caso não haja internet no final da checagem, a porcentagem não será enviada para o servidor.

Outro fator que pode prejudicar a checagem, é quando o participante passa muito tempo sem utilizar seu *smartphone*, o mesmo entra no modo de espera, onde a *CPU* e *WiFi* podem ser desabilitados, reduzindo o consumo de bateria. Mesmo a *API* utilizando *WakeLocks* para ativar a *CPU* e o *WiFi*, o dispositivo pareceu não reagir a isso.

A Tabela 4 apresenta os modelos dos *smartphones* e suas respectivas versões do Android.

Tabela 4 – Participantes e seus dispositivos

Participante	Modelo Smartphone	Versão do Android
Participante 1	Moto X	5.1
Participante 2	Asus Zenfone 5	5.1
Participante 3	Moto X	5.1
Participante 4	Moto G 2º Geração	5.0
Participante 5	Moto G 1º Geração	5.0

Outras versões foram utilizadas nos testes, sendo elas

- 5.0.1
- 4.0.3
- 4.2.2
- 6.0.0

Durante os teste efetuados na versão 6.0.0, percebeu-se que a API não possuia mais acesso aos dados das redes *WiFi*, mesmo requisitando as permissões *"android.permission.ACCESS_WIFI_STATE"* e *"android.permission.CHANGE_WIFI_STATE"*. Isso aconteceu por conta da nova mudança nas permições nas versões do Android superiores ou iguais a 6.0.0. Onde, o desenvolvedor deve requisitar a permissão do usuário em tempo de execução e a versão atual da API não dá suporte a esse tipo de evento.

6 CONSIDERAÇÕES FINAIS

Os ambientes *indoor* podem gerar grandes oportunidades para o desenvolvimento de novas aplicações, principalmente pelo fato das pessoas passarem maior parte de seus tempo nesse tipo de ambiente.

Esse trabalho apresentou uma API para auxiliar outros desenvolvedores na criação de novas aplicações para ambientes *indoor*. Com o desenvolvimento desse trabalho, ficou perceptível a aplicabilidade de pontos de acesso *WiFi* para localização em ambientes *indoor* desfrutando da vantagem de ser um equipamento barato.

Implementando uma aplicação e recolhendo os resultados, o funcionamento API da foi validado para a maior parte dos dispositivos testados.

Com os dados coletados em relação a variação da força do sinal recebido, pode-se concluir que a precisão do cálculo da distância fornecida pela API não possui uma acurácia grande. Sendo necessário, para aplicações mais refinadas, o uso de outra técnica de posicionamento como triangulação.

Ainda há bastantes pontos a serem melhorados na API. Um deles é verificar o consumo de bateria gerado pelo serviço de observação. Outro fator a ser observado é qual motivo levou a falha da checagem da presença no dispositivo do participante 5, ver Tabela 3. Para dá suporte as versões 6.0 em diante da plataforma Android, é necessário a implementação das permissões em tempo de execução.

Várias melhorias devem ser implementadas na aplicação de presença. Toda a comunicação entre ela e o servidor web está sendo feita sem nenhuma criptografia. Um teste com um número maior de participantes deve ser feito para observar o como o servidor web reage a uma carga maior de requisições.

REFERÊNCIAS

BRAHLER, S. Analysis of the android architecture. **Karlsruhe institute for technology**, v. 7, 2010.

CARVALHO, J. F. M. Localização de dispositivos móveis em redes wi-fi. 2007.

CONTE, E. et al. Algoritmos de análise de cena para localização indoor via redes ieee 802.11. 2015.

CRUZ, B. A.; LOVISOLO, L.; CAMPOS, R. S.; SZTAJNBERG, A. Um sistema para localização em tempo real de terminais wi-fi em ambientes indoor. **Cadernos do IME-Série Informática**, v. 31, p. 21–42, 2011.

FIGUEIREDO, C. M.; NAKAMURA, E. Computação móvel: Novas oportunidades e novos desafios. **Belo Horizonte: Universidade Federal de Minas Gerais: 28p**, 2003.

GAMMA, E. **Padrões de Projetos: Soluções Reutilizáveis**. [S.l.]: Bookman editora, 2009.

IPASS. **IPass Wi-Fi Growth Map**. 2016. Acesso: 2016-04-21. Disponível em: <http://www.ipass.com/wifi-growth-map>.

KIM, K.; CHA, H. Wakescope: runtime wakelock anomaly management scheme for android platform. In: IEEE PRESS. **Proceedings of the Eleventh ACM International Conference on Embedded Software**. [S.l.], 2013. p. 27.

LECHETA, R. R. **Google Android-3ª Edição: Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. [S.l.]: Novatec Editora, 2013.

LIU, H.; DARABI, H.; BANERJEE, P.; LIU, J. Survey of wireless indoor positioning techniques and systems. **Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on**, IEEE, v. 37, n. 6, p. 1067–1080, 2007.

MEIRELLES, F. d. S. 27ª pesquisa anual do uso de ti, 2016. 2016.

MIYABUKURO, E. Sistema de monitoramento de transporte coletivo em tempo real via gps para smartphone. 2015.

PEREIRA, C. R. **Aplicações web real-time com Node. js**. [S.l.]: Editora Casa do Código, 2014.

SIMÕES, D. M. **Navegação indoor baseada na rede WIFI como suporte a serviços baseados na localização: estudo de caso no campus da UL**. Tese (Doutorado), 2015.

VIRRANTAUS, K.; MARKKULA, J.; GARMASH, A.; TERZIYAN, V.; ALAINEN, J. V.; KATANOSOV, A.; TIRRI, H. Developing gis-supported location-based services. In: IEEE. **Web information systems engineering, 2001. Proceedings of the Second International Conference on**. Kyoto, Japão, 2001. v. 2, p. 66–75.

WEI-PING, Z.; MING-XIN, L.; HUAN, C. Using mongodb to implement textbook management system instead of mysql. In: IEEE. **Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on**. [S.l.], 2011. p. 303–305.

APÊNDICE A – Diagrama de Classe da API