UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

DIÓRGINO RIGLES ALVES REIS
MANUELY BARBOSA GUEDES

APLICATIVO MÓVEL PARA RASTREAMENTO

Estudo de Caso: Monitoramento do Ônibus Circular da UFPA

Belém - PA
2017

DIÓRGINO RIGLES ALVES REIS
MANUELY BARBOSA GUEDES

APLICATIVO MÓVEL PARA RASTREAMENTO
Estudo de Caso: Monitoramento do Ônibus Circular da UFPA

Trabalho de Conclusão de Curso apresentado para a obtenção do título de Bacharel em Sistema de Informação. Instituto de Ciências Exatas e Naturais. Faculdade de Computação. Universidade Federal do Pará.

Orientador: Prof. Dr. Dionne Cavalcante Monteiro

Belém - PA
2017

DIÓRGINO RIGLES ALVES REIS
MANUELY BARBOSA GUEDES

APLICATIVO MÓVEL PARA RASTREAMENTO
Estudo de Caso: Monitoramento do Ônibus Circular da UFPA

Trabalho de Conclusão de Curso apresentado para a obtenção do título de Bacharel em Sistemas de Informação. Instituto de Ciências Exatas e Naturais. Faculdade de Computação. Universidade Federal do Pará.
Belém, 10 de Outubro de 2017.

BANCA EXAMINADORA

_____

Prof. Dr. Carlos Gustavo Resque dos Santos

_____

Prof. Dr. Jefferson Magalhaes De Morais

_____

Prof. Dr. Dionne Cavalcante Monteiro

RESUMO

A funcionalidade de geolocalização, introduzida nos *smartphones*, se faz presente em uma série de aplicações capazes de identificar a localização de elementos de interesse ou definir trajetos eficientes para um deslocamento. Desta maneira, este trabalho apresenta o desenvolvimento de um aplicativo móvel para rastreamento, objetivando propor uma solução simples e acessível para o rastreamento de dispositivos acoplados a bens ou pessoas. O aplicativo é divido em dois módulos: Rastreamento, utilizado para obter as coordenadas de localização do dispositivo rastreado e Usuário, utilizado para acompanhar a movimentação de determinado dispositivo através do mapa. A aplicação foi desenvolvida em Android, utilizando API Google Maps, tecnologia GPS e tecnologia *webservice* o qual é responsável por intermediar a comunicação entre os módulos e a base de dados MySQL. Um Estudo de caso foi realizado dentro da Universidade Federal do Pará, através da utilização do aplicativo no monitoramento do Ônibus Circular. Sendo assim foi possível verificar o funcionamento real da aplicação bem como os benefícios que esta poderá trazer para a comunidade acadêmica.

Palavras-chave: Geolocalização; Rastreamento; Ônibus Circular; GPS; Aplicativo.

ABSTRACT

The feature of geolocation, introduced in smartphones, is present in a series of applications to identify the location of points of interest or to define efficient routes of movement . Therefore, this paper presents the development of a tracking mobile app, aiming to propose a simple and accessible solution for the tracking of devices attached to goods or people. The application is divided in two modules: Tracking, used to get the location coordinates of the tracked device, and User, used to track the movement of a particular device throughout the map. The application was developed in Android, using Google Maps API, GPS, and webservice technology, which is responsible for intermediating the communication between the modules and the MySQL database. A case study was conducted within the Federal University of Pará, by the use of the application in the monitoring of the Circular bus. Thus, it was possible to verify the actual operation of the application as well as the benefits that this could bring to the academic community.

Keywords: Geolocation; Tracking; Circular Bus; GPS; App.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE DIAGRAMAS

LISTA DE LISTAGENS

LISTA DE SIGLAS

AMPS	*Advanced Mobile Phone Service*
API	*Application Programming Interface*
CDMA	*Code Division Multiple Acess*
CDMA 1xEV-DO	*Evolution, Data-Optimized*
EDGE	*Enhanced Data Rates for Global Evolution*
eUMTS	*Evolved* UMTS
FDMA	*Frequency Divison Multiple Acess*
GPRS	*General Purpose Radio Services*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communication*
HAL	*Hardware Abstraction Layer*
HTTP	*Hypertext Transfer Protocol*
IDC	*International Data Corporation*
JDBC	*Java Database Connectivity*
JSON	*JavaScript Object Notation*
LTE	*Long Term Evolution*
MCS	*Japanese Mobile Cellular System*
NAMPS	*Narrowband AMPS*
NMT	*Nordic Mobile Telephone*
REST	*Representational State Transfer*
SOAP	*Simple object Access Protocol*
TAC	*Total Acess Communication System*
TCP	*Transmission Control Protocol*
UC	*Casos de Uso*
UML	*Unified Modeling Language*
UMTS	*Universal Mobile Telecommunications System*
URI	*Unified Resource Identifier*
WiMAX	*Worldwide Interoperability for Microwave Access*
WSDL	*Web Services Description Language*
W3C	*World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO

Quanto mais o homem evoluía, maior era a necessidade deste de se localizar e demarcar o território em que vivia. Para isso ele foi buscando maneiras que o auxiliasse em sua movimentação dentro e fora do espaço que ele ocupava. A princípio, a observação das estrelas, a criação de mapas e a bússola foram artifícios muito utilizados pelo homem para se orientar. Com a evolução da tecnologia, novas formas e ferramentas de localização foram desenvolvidas com o objetivo de facilitar e tornar mais precisa a orientação no espaço geográfico.

A popularização das tecnologias de geolocalização possibilitou o surgimento de inúmeras aplicações com o objetivo de disponibilizar informações a respeito da localização de pontos comerciais, pessoas, mercadorias, meios de transporte, bem como obter rotas e informações sobre o trânsito de determinado local. De acordo com Ferreira, Moreira e Mozzaquatro (2011), a geolocalização é importante por realizar a administração e procura reconhecer onde se encontram posicionadas as pessoas em determinado ponto geográfico.

Com o advento dos dispositivos móveis e a integração de funções, como: acesso à internet, e-mail e GPS (*Global Positioning System*). Os aplicativos de geolocalização tornaram-se mais acessíveis e presentes no cotidiano das pessoas.

O GPS, tecnologia presente em muitos *smartphones*, disponibiliza a localização de dispositivos de forma gratuita e com precisão de até 5 metros os aparelhos mais recentes (UOL Notícias, 2017). A rapidez na coleta de dados de posicionamento permite que aplicações possam obtê-los em tempo ágil e, assim, fornecer informações úteis ao usuário.

O Sistema Operacional Android domina, atualmente, 87,6% do mercado de *smartphones*, segundo o relatório do *International Data Corporation* (IDC) publicado em agosto de 2016. Ele oferece suporte ao uso do GPS, possibilitando o uso massivo deste serviço. A Google, como proprietária deste sistema, disponibiliza uma série de API's (*Application Programming Interface*) integradas, que podem ser utilizadas nos mais diversos tipos de aplicações. Uma das principais API's oferecidas pela Google é a Google Maps, que, atualmente, está em sua versão 3.0. A API permite o uso de recursos avançados de mapeamento, onde é possível exibir um ou mais locais em um mapa interativo.

Desta maneira, foi desenvolvido um aplicativo móvel para rastreamento, que utiliza as tecnologias mencionadas em sua implementação, provendo uma maneira intuitiva, simplificada e acessível de acompanhar a movimentação de dispositivos sendo estes acoplados a pessoas ou bens.

No trabalho em questão a aplicação será utilizada no contexto da Universidade Federal do Pará, através do monitoramento do Ônibus Circular, por meio do aplicativo aqui desenvolvido, onde a utilização do mesmo pode dar um ganho significativo de tempo as pessoas que utilizam esse meio de transporte.

1.1. Justificativa

Atualmente em nossa sociedade, é notória a necessidade que as pessoas sentem em obter informações em tempo real da localização de objetos de seu interesse, sejam carros, ônibus, motos ou pessoas. Como forma de atender a essa necessidade, emergiu o interesse em desenvolver um aplicativo com fácil aplicabilidade e acessível à comunidade para o recebimento dessas informações.

Partindo dessa premissa e com a popularização dos smartphones, este estudo visa fomentar a criação de um aplicativo que monitore e mostre em tempo real a localização de qualquer objeto. Em consonância a criação do aplicativo, buscou-se utiliza-lo como uma ferramenta que possa vir auxiliar a comunidade acadêmica da UFPA, através da disponibilização de informações a respeito da localização do Ônibus Circular dentro da Universidade, pois devido a grande extensão da mesma, e sem nenhuma ferramenta que disponibilize tais informações a respeito do mesmo, o tempo de espera dos usuários por este meio de transporte acaba aumentando.

Com o uso deste aplicativo, a comunidade acadêmica e demais usuários, poderão ser beneficiados, pois de posse das informações de localização do ônibus poderão desfrutar da melhor maneira deste importante recurso de locomoção disponibilizado pela Universidade.

1.2. Objetivo Geral

Este trabalho tem por objetivo geral desenvolver um aplicativo móvel que possibilite realizar o rastreamento de dispositivos, sendo este aplicado ao contexto da Universidade Federal do Pará, através do monitoramento do Ônibus Circular.

1.3. Objetivos Específicos

- Disponibilizar uma solução simples e acessível para o monitoramento da posição do ônibus Circular.
- Exibir a Localização do Circular em Mapa através de aplicativo móvel.

- Verificar os benefícios que a comunidade acadêmica pode obter com a utilização da aplicação.

1.4. Materiais e Métodos

Inicialmente, o trabalho buscou o levantamento bibliográfico acerca das tecnologias envolvidas. Posteriormente, foi feita a leitura das literaturas mais relevantes, reforçando o aprendizado do assunto. Ao final, houve estudo dos conceitos gerais e análise de documentações.

Em seguida, iniciou-se a fase de modelagem do projeto. Usou-se, predominantemente a UML (*Unified Modeling Language*) compreendida como a Linguagem de Modelagem Unificada para representar os módulos e componentes do sistema. Nesta etapa, foram aplicados conceitos fundamentais da orientação a objetos.

Como forma de segmentar as implementações e facilitar o entendimento, o projeto foi dividido em duas partes: a primeira, *o Web Service* e a segunda, o aplicativo móvel. Cada uma contém uma especificação de casos de uso e diagramas representativos.

Em seguida, foram levantadas as alternativas tecnológicas existentes, a fim de identificar com quais os desenvolvedores possuem maior afinidade, em intersecção com as que sejam mais viáveis para o perfil de aplicação que se pretende chegar. A partir deste processo, optou-se pelo uso da plataforma Android e da linguagem Java, por ser a linguagem padrão para desenvolvimento nesta plataforma. As demais tecnologias foram inseridas posteriormente, conforme a necessidade apresentada.

De posse da documentação inicial e do ambiente de desenvolvimento devidamente configurado, iniciou-se o desenvolvimento da aplicação a partir de um projeto vazio (não foram utilizados *templates*).

A primeira parte a ser desenvolvida foi o *Web Service*, o qual visa, exclusivamente, atender às requisições do aplicativo móvel, por meio de chamadas assíncronas. Seu papel é intermediar a persistência em base de dados, conforme os dados fornecidos pelos aplicativos clientes. Desenvolvidos os serviços necessários, testes unitários foram desempenhados sobre cada um, verificando a correção das regras de negócio implementadas no *Web Service*.

A segunda parte foi o aplicativo móvel, que contém todas as funcionalidades disponibilizadas. Devido aos recursos utilizados na aplicação, o nível mínimo de API suportado é o *Jelly Bean* (versão 4.1), atingindo a maior parte dos usuários deste sistema operacional.

Após o desenvolvimento das duas partes do sistema, testes locais foram realizados para verificar a integração entre ambas. Feitos ajustes, o *Web Service* foi disponibilizado na internet para viabilizar testes de campo.

Nas etapas de teste, o desempenho das tecnologias foi avaliado quanto a consumo de recursos de hardware (análise de travamentos dos dispositivos utilizados), desempenho (tempo de resposta às requisições) e portabilidade (execução em diferentes dispositivos com Android).

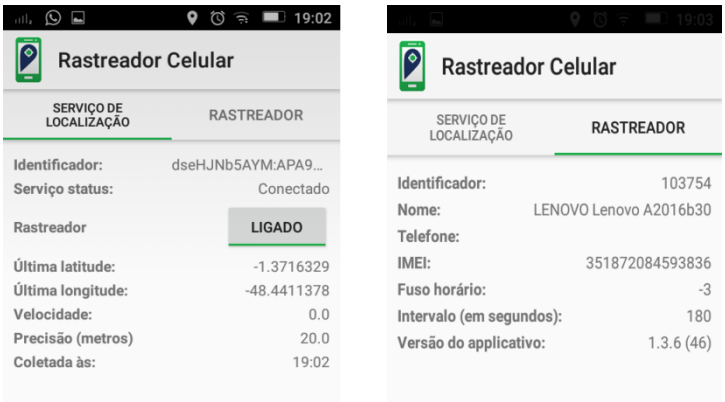
1.5. Trabalhos Correlatos

Nesta seção serão descritos quatro aplicativos móveis que possuem finalidade similar ao do aplicativo desenvolvida neste trabalho.

1.5.1. Rastreador Celular

O Rastreador Celular é um aplicativo móvel para rastreamento de *smartphones* e *tablets* desenvolvido pela empresa MobilTracker. De acordo com o site da empresa, com esse tipo de rastreamento é possível rastrear equipes de vendas, entregadores, acompanhar a localização de um familiar e até mesmo proteger o aparelho de possíveis roubos.

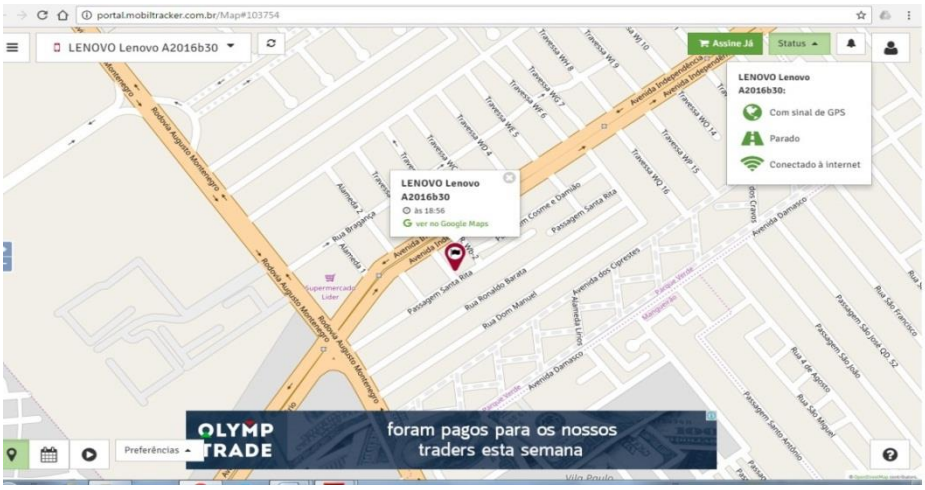
Figura 1 - Telas Rastreador Celular



Fonte: Aplicativo Rastreador Celular

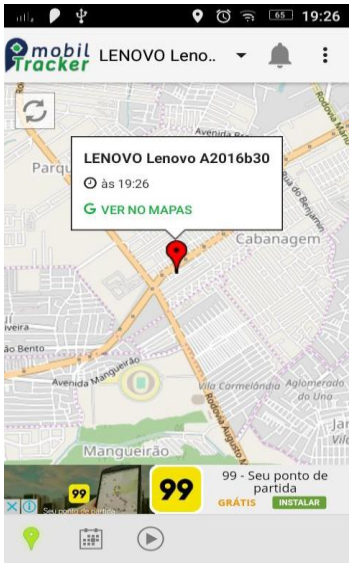
Na plataforma web e móvel disponibilizada pela empresa, o usuário pode visualizar a localização em mapa do dispositivo rastreado, seu histórico de localização e até enviar comandos via SMS para bloquear ou configurar o dispositivo rastreado.

Figura 2 - Tela do PortalWeb do Rastreador Celular



Fonte: http://portal.mobiltracker.com.br/Map#103754

Figura 3 – Tela Aplicativo MobilTracker Monitor



Fonte: Aplicativo MobilTracker Monitor

Para utilizar o sistema é necessário criar uma conta acessando o Portalweb da empresa. É possível obter uma conta gratuita, que dá direito ao rastreamento de apenas um dispositivo, mas a empresa disponibiliza planos pagos, que permitem o monitoramento de mais de um dispositivo.

1.5.2. SIRASS

Em trabalho semelhante, Kieltyka (2013), desenvolveu o SIRASS, que é um sistema para rastreamento de *smartphones*. Tal sistema permite o rastreamento de dispositivos móveis através de um aplicativo instalado no mesmo. De modo geral ele é constituído de três módulos:

- Aplicativo móvel: responsável pelo envio dos dados do usuário e da localização do *smartphone*.
- *Web Service*: oferece todos os serviços utilizados pelo aplicativo e pelo site web, como buscar na base de dados informações relacionadas ao usuário e a localização atual do *smartphone*.
- Site de Busca: permite consultar e visualizar em mapa a localização de um *smartphone* em tempo real, bem como localizações anteriores.

Figura 4 - Tela Aplicativo SIRASS

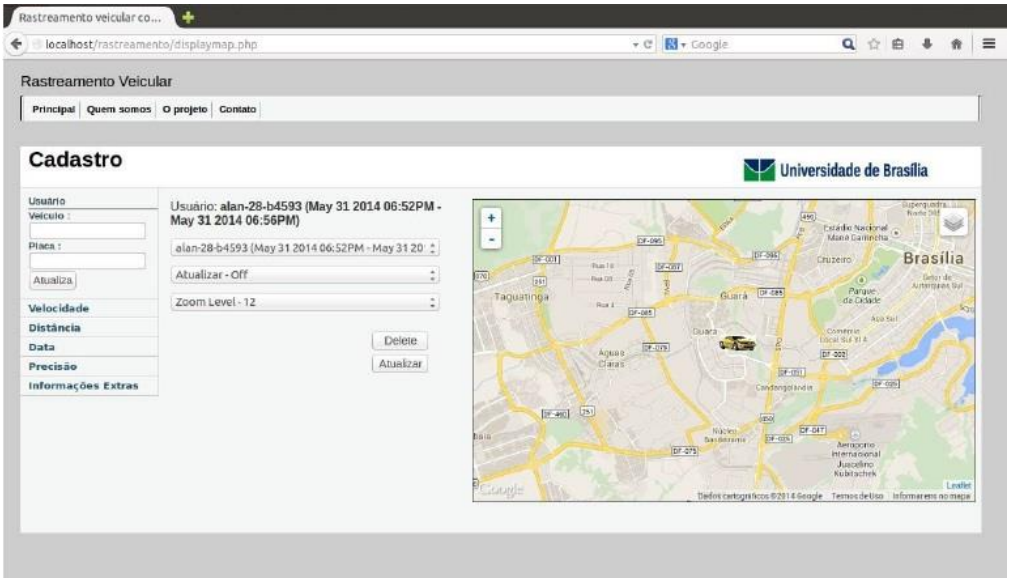
Fonte: Kieltyka (2013)

1.5.3. Rastreamento Veicular com auxílio de dispositivos móveis

Em trabalho correlato, Ferreira e Gonçalves (2014) apresentam o desenvolvimento de um sistema de rastreamento veicular cujo objetivo é auxiliar os usuários no rastreio remoto de eventuais furtos e/ou roubos de seus veículos.

O sistema utiliza um dispositivo móvel dentro do veículo para realizar a coleta das informações de localização e envio das mesmas a um servidor. Os usuários fazem a consulta dessas informações através de um website, o qual dispõe de um mapa digital que permite a visualização da localização do veículo.

Figura 5 - Tela Site para Consulta do Sistema Rastreamento Veicular



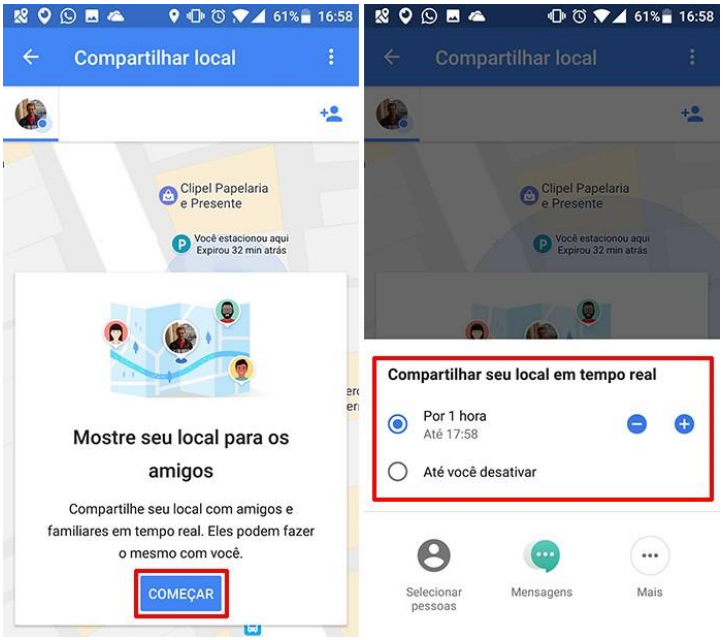
Fonte: Ferreira; Gonçalves (2014)

1.5.4. Compartilhamento de Localização GoogleMaps

O GoogleMaps é um serviço de mapa online desenvolvido e disponibilizado gratuitamente pela empresa Google. Com esta ferramenta é possível pesquisar por lugares específicos (bares, restaurantes, academias, entre outros), obter rotas para auxiliar no deslocamento de um lugar a outro e conhecer diferentes locais do mundo através de imagens com vista panorâmica, disponíveis através do recurso de *Street View*.

Além dos recursos mencionados, o GoogleMaps conta com um novo serviço lançado em março de 2017, o de compartilhamento de localização. Os usuários da ferramenta podem compartilhar sua localização em tempo real com seus familiares e amigos. O recurso permite que o usuário escolha com quem irá compartilhar sua localização e por quanto tempo, o qual pode variar de quinze minutos a três dias, sendo possível interromper o compartilhamento a qualquer momento. Dessa maneira, é possível acompanhar no mapa o deslocamento de determinado usuário, desde que haja permissão para tal.

Figura 6 - Compartilhamento Localização GoogleMaps



Fonte: http://www.techtudo.com.br/dicas-e-tutoriais/noticia/2017/04/google-maps-como-compartilhar-sua-localizacao-em-tempo-real-no-android.html

1.5.5. Comparação

Os trabalhos correlatos apresentados têm relação com o aplicativo desenvolvido neste trabalho, denominado RastreWorld, pelo fato dos mesmos apresentarem soluções para rastreamento de bens ou pessoas utilizando dispositivos móveis. Porém se diferem em relação as suas funcionalidades, as quais estão descritas na Tabela 1.

Tabela 1 - Comparação de funcionalidades entre os sistemas

	Rastreador Celular	SIRASS	Rastreamento Veicular	GoogleMaps	RastreWorld (Proposto)
Consulta localização Via Aplicativo Móvel	Sim	Não	Não	Sim	Sim
Consulta localização Via Website	Sim	Sim	Sim	Sim	Não
Dispositivos rastreados por um mesmo Usuário	1 (Versão Gratuita)	Apenas 1	Apenas 1	Mais de 1	Mais de 1
Versão Gratuita	Sim	Sim	Sim	Sim	Sim
Versão Paga	Sim	Não	Não	Não	Não
Alerta de Aproximação	Não	Não	Não	Não	Sim

Fonte: Elaborado pelos autores

O aplicativo Rastreador de Celular e o recurso de Compartilhamento de Localização do GoogleMaps tem por vantagem disponibilizar a consulta de localização dos dispositivos rastreados, tanto pela web como via aplicativo móvel, porém o Rastreador de celular permite o rastreio de um único dispositivo em sua versão gratuita, ao contrário do GoogleMaps, em que vários dispositivos podem ser rastreados simultaneamente.

Já o SIRASS e o Rastreamento Veicular, permitem a consulta de localização dos dispositivos somente por meio de website. Além de realizar o rastreamento de apenas um dispositivo por usuário.

Sendo assim, o aplicativo RastreWorld, apesar de não possuir uma plataforma web para a consulta de localização de dispositivos rastreados, apresenta como vantagem a possibilidade de um usuário poder rastrear diversos dispositivos de maneira gratuita, bem como a facilidade de consulta via aplicativo móvel, o que provê maior comodidade, permitindo o rastreamento de itens a qualquer momento e local. Outra vantagem em relação aos demais, é que ele oferece ao usuário uma funcionalidade que notifica o mesmo, através de alerta sonoro e vibração do dispositivo quando determinado objeto/dispositivo rastreado está se aproximando deste.

1.6. Estrutura do Trabalho

Este trabalho está dividido em seis capítulos distribuídos da seguinte maneira;

- Capítulo 1: descreve os objetivos, justificativa e metodologia adotada na construção da aplicação.
- Capítulo 2: apresenta a fundamentação teórica abordando os conceitos das tecnologias utilizadas para desenvolver a aplicação.
- Capítulo 3: é realizada a descrição do funcionamento do aplicativo, seus requisitos e modelagem do sistema.
- Capítulo 4: trata dos detalhes da implementação do aplicativo Android e do *Web Service*.
- Capítulo 5: descreve como foram realizados os testes da aplicação e dos resultados obtidos a partir dele.
- Capítulo 6: expõe a conclusão geral sobre o trabalho, apresentando uma análise acerca da aplicação desenvolvida.

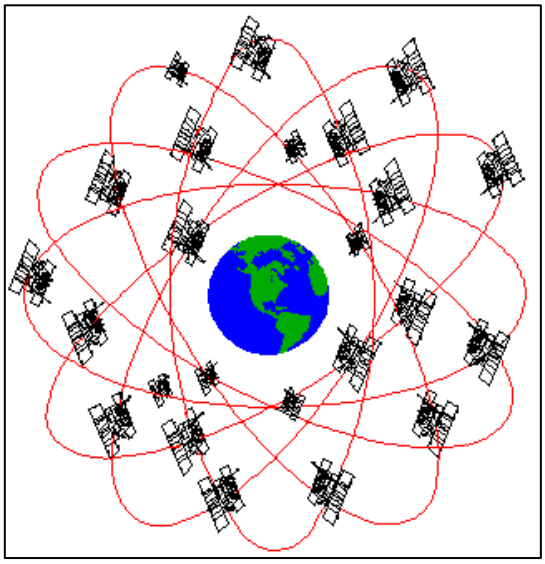
2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados conceitos referentes às tecnologias utilizadas para o desenvolvimento deste projeto. Sistema de Posicionamento Global, sistema operacional Android e tecnologia *Web Service* são algumas dessas tecnologias. Linguagem de programação e rede de transmissão de dados também serão abordados.

2.1. Sistema de Posicionamento Global (GPS)

Sousa (2005) contextualiza que o Sistema de Posicionamento Global, mais conhecido pela sigla GPS, é um sistema de posicionamento baseado em satélites que foi criado na década de 70 pelo Departamento de Defesa dos Estados Unidos. Inicialmente foi desenvolvido para fins militares, e posteriormente disponibilizado para o uso civil. A ideia de funcionamento do sistema GPS permite ao usuário determinar sua posição em qualquer lugar do planeta, em tempo real. Sendo isto possível devido aos 24 satélites distribuídos ao redor da Terra, conforme Figura 7.

Figura 7 - Distribuição de Satélites ao redor da Terra



Fonte: Nós e a Química (2008)

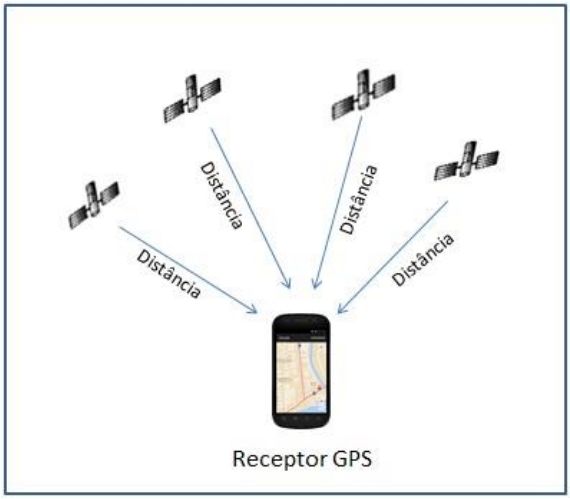
Para utilizar o sistema GPS, o usuário necessita de um equipamento para receber e decodificar os sinais enviados pelos satélites que compõem o sistema. Esses equipamentos são denominados receptores GPS. No mercado atual é possível encontrar uma grande variedade desses equipamentos, os quais se diferenciam em custo e funcionalidades (MARQUES, 2006). A utilização desse sistema se tornou algo muito presente no cotidiano das pessoas, a partir do momento em que os receptores GPS foram integrados a celulares e *smartphones*, possibilitando o desenvolvimento de aplicações como a que está sendo descrita neste trabalho.

Segundo estudo realizado por Marques (2006, p.15) pondera-se que:

> O princípio básico para o funcionamento de um sistema GPS se deve a medição da distância entre a antena do receptor e as antenas dos satélites da constelação. A partir da medição desta distância entre a antena do receptor e de quatro satélites ao mesmo tempo, pode-se determinar a posição geográfica (coordenada) da antena do receptor.

A Figura 8 representa este processo.

Figura 8 - Processo de determinação de posição por GPS.



Fonte: Elaborado pelos autores

Os sistemas de posicionamento baseados em satélites estão sujeitos à possibilidade de perda de sinal na presença de altos edifícios, túneis e viadutos impossibilitando o cálculo da posição do receptor (SILVA, 2006).

2.2. Protocolo HTTP

O protocolo de Transferência de Hipertexto (HTTP) é um protocolo da camada de aplicação da pilha TCP/IP, que utiliza uma arquitetura cliente-servidor para realizar a comunicação entre dois sistemas finais diferentes, sendo esta comunicação efetuada por meio de trocas de mensagens HTTP (KUROSE e ROSS, 2010, p. 74).

O HTTP define a estrutura dessas mensagens e o modo pelo qual clientes e servidores devem trocá-las. O HTTP utiliza o protocolo TCP (*Transmission Control Protocol)* traduzindo para Protocolo de Controle de Transmissão, servindo para transferir os dados entre cliente e servidor (KUROSE e ROSS, 2010). Primeiramente o cliente estabelece uma conexão TCP com o servidor, em seguida os processos de ambos acessam o TCP por meio de suas interfaces *sockets*. Clientes e servidores utilizam suas interfaces *sockets* tanto para enviar mensagens de requisição HTTP quanto para receber mensagens de resposta (TANENBAUM, 2010).

Conforme classificam Kurose e Ross (2010), o protocolo HTTP pode atuar com dois tipos de conexão:

- Persistente: A aplicação cliente estabelece uma conexão TCP com o servidor e solicita todos os objetos desejados, sendo a conexão encerrada após o envio de todos os objetos.
- Não persistente: A aplicação cliente estabelece uma conexão TCP com o servidor para cada objeto desejado e após o recebimento do objeto, a conexão é encerrada. Para o envio de outro objeto, abre-se uma nova conexão. Isso é feito até a conclusão do envio de todos os objetos.

2.3. Rede Móvel de Transmissão

Redes móveis de transmissão são infraestruturas de comunicação que surgiram em 1980. No início a rede oferecia somente o serviço básico de transmissão de voz, permitindo ao usuário receber ou realizar chamadas de aparelhos não fixos (LÁRIOS, 2003). Projetos como TACS (*Total Access Communication System*), o NMT (*Nordic Mobile Telephone*), o NAMPS (*Narrowband* AMPS) e o AMPS (*Advanced Mobile Phone Service*), são exemplos dos primeiros sistemas de transmissão móvel desenvolvidos (PAULA; BRESSAN; AGE, 2013).

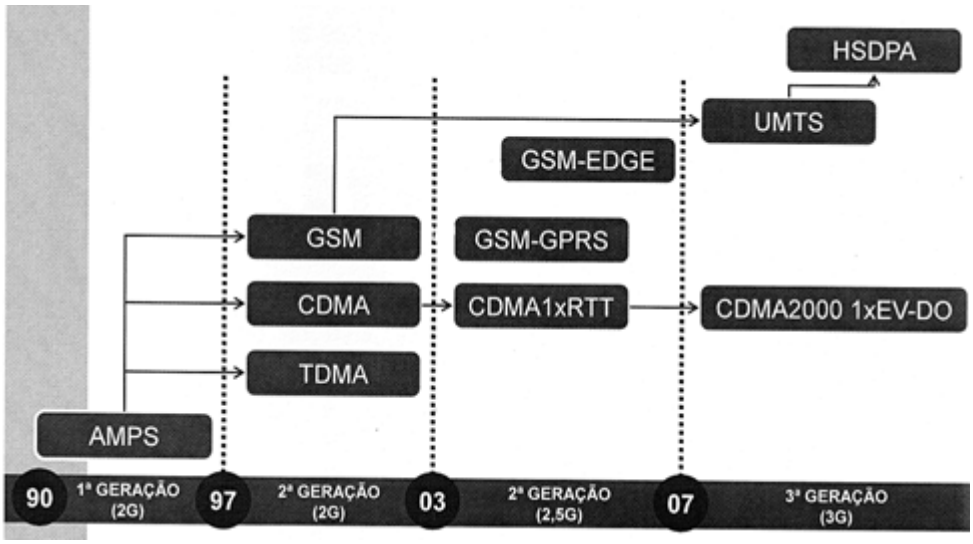
A princípio as redes móveis tinham por objetivo ter uma área abrangente de comunicação a partir de apenas um transmissor, utilizando a tecnologia, FDMA (*Frequency Division Multiple Access*), cada usuário possuía uma frequência específica, apesar de ter uma boa cobertura, a capacidade de usuário era bem baixa, tornando óbvia a necessidade de melhorias (ALMEIDA, 2013).

Com o aumento da grande demanda de telefonia móvel na década de 90 e o avanço nas tecnologias dos circuitos integrados, as empresas desenvolveram as tecnologias digitais como CDMA200 (*Code Division Multiple Acess*) e o GSM (*Global System for Mobile Communication*), que implementaram a criptografia nas chamadas, melhor qualidade nas ligações e tráfego de dados. A segunda geração possui uma segunda fase que serviu para aumentar significativamente a taxa de transferência, sistemas *Enhanced Data Rates for Global Evolution* (EDGE) e o *General Purpose Radio Services* (GPRS) foram os responsáveis por esse avanço.

A terceira geração possui duas grandes tecnologias UMTS (*Universal Mobile Telecommunications Syste*m) e a CDMA 1xEV-DO (*Evolution, Data-Optimized*), essa nova

geração trouxe a banda larga as redes móveis, taxas de transferências de 2g, além de ter uma menor interferência.

Figura 9 - Evolução da Tecnologia Móvel



Fonte: Silva (2010)

Estamos atualmente na quarta geração, são as tecnologias LTE (*Long Term Evolution* - 3GPP R8 e R9), também chamadas de eUMTS (*Evolved* UMTS) e WiMAX (*Worldwide Interoperability for Microwave Access* – IEEE 802.16) possui taxa inicial de 150Mbps, mas que atualmente atinge 300Mbps e até 1Gbps.

A tecnologia móvel evolui tanto que está chegando ao ponto de substituir funções antes apenas restritas a conexões fixas, como navegação na internet, email, redes sociais, e principalmente as aplicações que necessitam de elevadas taxas de transmissão como, por exemplo, a transmissão de vídeos em alta definição e/ou em tempo real.

2.4. Linguagem Java

Inicialmente o Java se chamava “Oak”, que foi trocado em 1995 para Java devido já existir uma linguagem com esse nome, o objetivo era ser uma linguagem específica para processadores de eletrodomésticos, já que eletroeletrônico é um ramo que evolui significativamente, no entanto, os softwares que existiam não conseguiam acompanhar ao mesmo passo essa evolução, sendo necessário sempre um novo programa a cada novo processador, o primeiro projeto foi o Projeto Green, que exibia e controlava em um computador manual (nomeado de star seven) os aparelhos da casa (SILVEIRA, 2003).

Em 1993 a linguagem “Oak” não estava sendo bem aceita, por ter um custo muito alto para que os aparelhos suportassem a nova linguagem, no entanto a *world wide web* estava no seu início, e os desenvolvedores da *Sun* ao ver o navegador *mosaic* no mercado, perceberam

que a linguagem seria perfeita para aplicação web, já que poderia rodar em qualquer tipo de computador ligado a internet.

Java é uma linguagem de alto nível voltada para programação orientada a objetos utilizada nos mais diversos tipos de plataformas (web, aplicativos móveis, desktop), possui uma linguagem bem simples e muito similar a C++ além de ser fortemente tipada, independente de arquitetura, robusta, segura, extensível, bem estruturada, distribuída e multithread, é atualmente mantida pela Oracle e se encontra na versão 8.

2.5. Sistema Operacional Android

O sistema operacional Android, teve seu desenvolvimento iniciado pela Android Inc., uma empresa de pequeno porte localizada em Palo Alto, Califórnia. Posteriormente esta foi adquirida pela Google, a qual estava iniciando seu caminho em direção ao mercado mobile. Em 2007, surgiu a *Open Handset Alliance*, um grupo de empresas que se reuniram com o intuito de acelerar a inovação em dispositivos móveis, trazendo uma melhor experiência para os consumidores através do desenvolvimento de uma plataforma completa, aberta e gratuita (MONTEIRO, 2012).

Segundo Lecheta (2010), o Android é uma plataforma para desenvolvimento de aplicações para dispositivos móveis, que inclui um sistema operacional baseado no Linux, tendo uma interface visual rica, diversas aplicações já instaladas e um ambiente de desenvolvimento poderoso e flexível.

A plataforma Android permite desenvolver aplicações que utilizam os mais diversos tipos de recursos de hardware do celular. Ela oferece classes nativas que permite o uso do GPS, câmera fotográfica, lista de contatos, entre outros. Além disso, é possível integrar as aplicações com os mais diversos serviços oferecidos pela Google, como contas do Gmail e GoogleMaps.

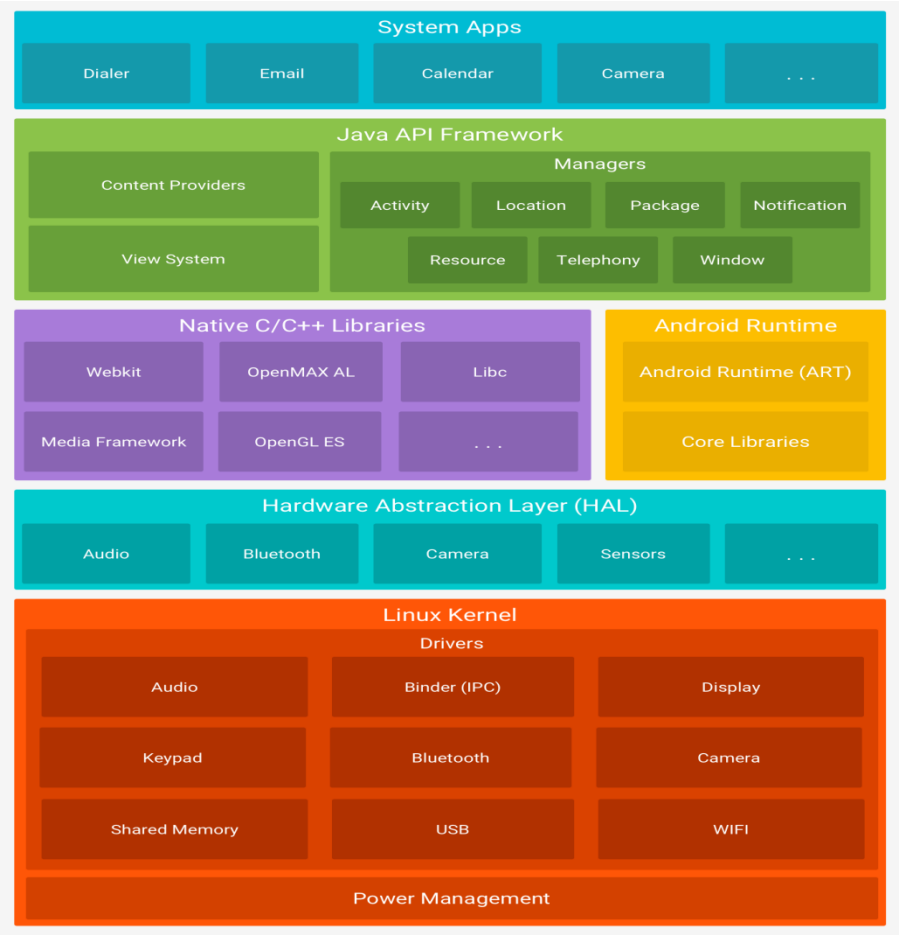
2.5.1. Arquitetura Android

A arquitetura da plataforma Android, segundo o site Android Developer (2017), é constituída de uma pilha de software. Conforme a Figura 10, na base dessa pilha está à camada do Kernel, que é baseada no sistema operacional Linux, sendo esta responsável pelos principais serviços do sistema Android, como gerenciamento de memória e processos.

Logo acima, está a camada HAL (*Hardware Abstraction Lay*er), trata-se de uma camada que tem por objetivo fornecer uma abstração entre o hardware do dispositivo e seus aplicativos, minimizando dessa maneira diferenças entre drives de diferentes dispositivos.

Para isso, ela conta com um conjunto de bibliotecas que implementam uma interface para um determinado tipo de hardware, como o módulo de câmera ou *bluetooth*.

Figura 10 - Arquitetura Android



Fonte: https://developer.android.com/guide/platform/index.html (2016)

A camada de bibliotecas nativas fornece funcionalidades que permitem a utilização de recursos como banco de dados e manipulação de gráficos. Um exemplo dessas bibliotecas é a SQLite, utilizada para acessar e manipular banco de dados dentro de uma aplicação. Grande parte das bibliotecas nativas foi desenvolvida em C e C++, dessa forma, a plataforma Android oferece a Java Framework API's, para expor as funcionalidades de algumas dessas bibliotecas aos aplicativos (ANDROID DEVELOPES, 2016).

O Android *RunTime* é constituído por um conjunto de bibliotecas que fornecem várias funcionalidades da linguagem Java e por uma maquina virtual, a ART (*Android RunTime*), responsável pela compilação e execução dos aplicativos no sistema Android, sendo que cada aplicativo roda em seu próprio processo. Em versões anteriores ao Android 5.0 (API nível 21), esse papel é realizado pela máquina virtual *Dalvik* (ANDROID DEVELOPES, 2016).

Na camada de Framework estão disponíveis através de API's Java todos os recursos necessários para o desenvolvimento de aplicações para o sistema Android. Os desenvolvedores tem acesso às mesmas API's que os aplicativos nativos do Android utilizam.

No topo dessa pilha de software encontra-se a camada de aplicativos do sistema, composta pelo conjunto de aplicativos que já vêm inclusos na plataforma. Dentre os quais, destacam-se aplicativos de e-mail, jogos, navegador de internet, contatos, mapa entre outros.

2.6. Web Service

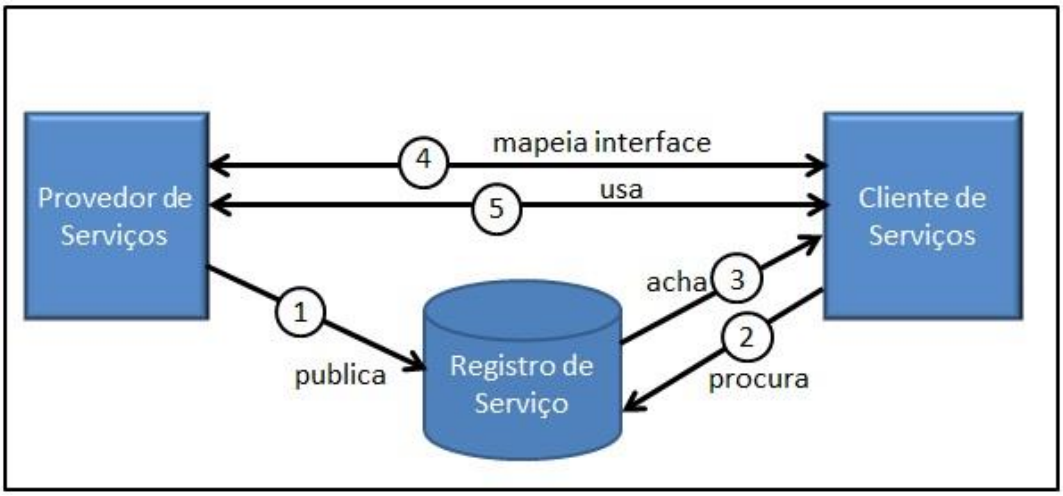
O W3C (*World Wide Web Consortium*) define um *Web Service* como um sistema de software projetado para permitir a integração entre diferentes aplicações executando em diferentes plataformas através de uma rede (W3C, 2004). O qual possui uma interface descrita em um formato processável, especificamente WSDL (*Web Services Description Language*). Outros sistemas interagem com *Web Service* através do uso de mensagens do tipo SOAP (*Simple Object Access Protocol*) ou outro protocolo, que normalmente são transmitidas utilizando o HTTP (RODRIGUES, 2009).

O WSDL é uma especificação W3C, que descreve o conteúdo das mensagens trocadas, onde o serviço está disponível e quais protocolos de comunicação são utilizados para acessar o serviço (LIMA, 2012).

Segundo Rodrigues (2009), a arquitetura *Web Service* é baseada na integração de três elementos ou entidades: Provedor de Serviços, Cliente do Serviço e Registro de Serviço.

- Provedor de Serviço: Responsável pela implementação e disponibilização dos *Web services*, em um formato padrão e compreensível a qualquer cliente que deseje utilizá-lo.
- Cliente do Serviço: é qualquer aplicação que deseje utilizar um serviço web disponibilizada por um provedor. Podendo ser uma pessoa acessando através de um browser ou uma aplicação realizando uma chamada aos métodos descritos na interface do *Web Service*.
- Registro de Serviço: Especifica a localização do serviço, além de conter as informações técnicas sobre o mesmo.

Figura 11 - Interação entre os elementos da arquitetura *Web Service*



Fonte: Adaptado de Lima (2012)

A Figura 11 demonstra a interação entre os elementos da arquitetura *Web Service*, onde o provedor de serviço publica a descrição de um serviço para os clientes no registro de serviços. Dessa maneira os clientes buscam por serviços nos servidores de registro, e se comunicam com os provedores para interagir com a implementação do *Web Service.*

2.6.1. REST

REST (*Representational State Transfer*) é um estilo de desenvolvimento de *Web Services* que teve sua origem na tese de doutorado de Roy Fielding, na qual este buscou as melhores práticas nos estilos arquiteturais já existentes para compor um novo estilo, que ficou conhecido como REST (LIMA, 2012 apud FIELDING, 2000).

Segundo Saudate (2014) em REST, tudo é definido em termos de recursos. Um recurso pode ser um documento, uma imagem ou mesmo um serviço. Sendo que cada recurso deve ter uma identificação única o qual é representado por meio de URI (*Unified Resource Identifier*). Outras características que podem ser destacas desse modelo de *Web Service* são:

- **Ações**: várias operações podem ser realizadas sobre um recurso utilizando os diferentes métodos disponibilizados pelo HTTP. Por exemplo, o método GET (Recupera o conteúdo de um recurso), PUT/POST (altera ou cria o conteúdo do recurso) e DELETE (apaga o conteúdo do recurso), citando os mais usados (EGGEA, 2013).
- **Media types**: utilizado para definir em que formato de dados o conteúdo do recurso deve ser representado para trafegar pela rede. Dentre os formatos de dados que podem ser utilizados pela abordagem REST estão o JSON e o XML (SAUDATE, 2014).

- **Comunicação Sem Estado**: nesta abordagem o servidor não armazena nenhuma informação. Dessa forma, todas as requisições realizadas pelo cliente devem conter as informações necessárias para que ela possa ser processada pelo servidor (RODRIGUES, 2009).

Como já mencionado, um recurso pode ser um documento, uma imagem, a representação de um objeto físico. Para acessar tal recurso é necessário associá-lo a um nome e a um local onde ele possa ser encontrado. Em REST, isto se dá por meio de URI's.

As URI's devem ser definidas de maneira intuitiva para que fiquem claros para os desenvolvedores quais são os recursos presentes na aplicação e os métodos HTTP que podem ser utilizados com a mesma. O Quadro 1 mostra um exemplo de URI's definidas para um recurso "usuário", bem como os métodos HTTP que podem ser utilizados.

Quadro 1 – Exemplo de URI's em REST

URI	Método HTTP
/usuario	GET (recupera todos os usuários)
	POST (cria um novo usuário)
	DELETE (excluir todos os usuários)
/usuario/{id}	GET (recupera determinado usuário)
	PUT (altera os dados de determinado usuário)
	DELETE (exclui determinado usuário)

Fonte: Elaborado pelos autores

Por possuir as características mencionadas, o modelo REST foi adotado para construir o *Web Service* deste projeto. Desta maneira, serão utilizados os métodos do protocolo HTTP para realizar as operações de CRUD nos recursos definidos no projeto, sendo que cada recurso deve possuir sua URI específica, para que este possa ser acessado pelo cliente por meio de requisições HTTP. O formato de dados JSON será utilizado para codificar a resposta a uma requisição do cliente, bem como para enviar dados necessários para o processamento da mesma.

3 PROJETO DE SOFTWARE

Neste capítulo será apresentada uma visão geral do funcionamento do aplicativo desenvolvido; e detalhes da especificação do mesmo como, arquitetura, requisitos e modelagem, bem como a descrição de suas funcionalidades.

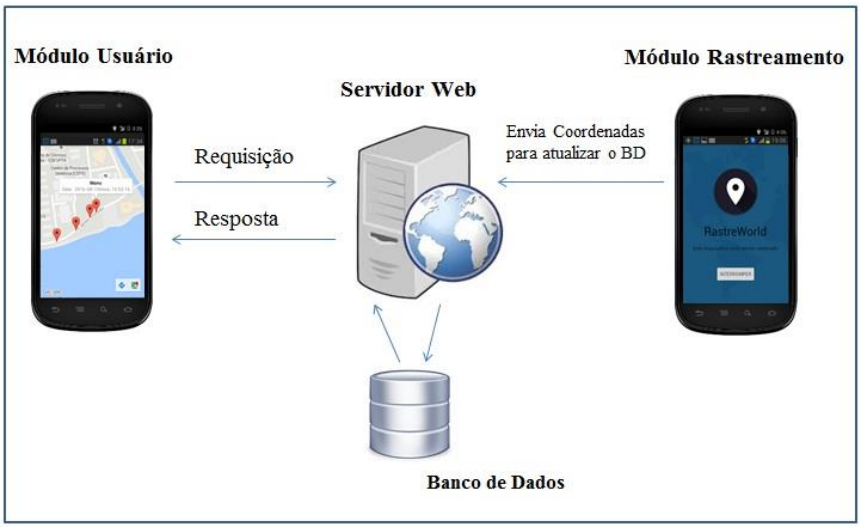
3.1. Visão Geral

Neste trabalho foi desenvolvido um aplicativo móvel para rastreamento de dispositivos, o qual será atrelado ao Ônibus Circular da UFPA, permitindo assim, acompanhar a movimentação deste por meio de mapas. Apesar deste projeto realizar um estudo de caso com o monitoramento do Ônibus Circular da UFPA, o aplicativo desenvolvido pode ser utilizado para rastrear qualquer elemento que possa ser atrelado a um *smartphone*, dessa maneira o dispositivo a ser rastreado será chamado de objeto. O aplicativo possui dois módulos bem definidos, o Módulo Rastreamento**,** responsável pela captura das coordenadas de localização do dispositivo/ônibus e envio das mesmas a um servidor por meio de *Web Service* para armazenar em uma base de dados e o Módulo Usuário responsável pela visualização em mapa do deslocamento do objeto/ônibus que está sendo rastreado. O aplicativo faz automaticamente as solicitações das coordenadas para o *Web Service* e, após receber a resposta, adiciona um marcador no mapa, representando a atual posição do objeto/ônibus.

Para obtenção de informações referente a usuários e objetos cadastrados, foi criado um servidor web, que permite a comunicação do aplicativo com a base de dados. A Figura 12 representa o funcionamento do sistema.

É importante ressaltar que apesar da divisão do aplicativo em módulos, este é um aplicativo único, no qual é possível definir qual o módulo será utilizado pelo dispositivo no qual foi instalado. Também é importante mencionar que não é possível utilizar os dois módulos ao mesmo tempo em um mesmo dispositivo.

Figura 12 - Funcionamento



Fonte: Elaborado pelos Autores

3.2. Especificações

Nesta seção são apresentados os requisitos funcionais da aplicação, bem como sua arquitetura e representação de classes e casos de uso através de diagramas UML.

3.2.1. Requisitos Funcionais

Os requisitos funcionais são requisitos que especificam quais operações um sistema deve ser capaz de executar (FURTADO e COSTA JUNIOR, 2010). A Tabela 3 descreve os requisitos funcionais do aplicativo.

Tabela 2 – Requisitos Funcionais

Nº	**Descrição**
RF01	O sistema deve registrar o login do usuário assim como seus dados pessoais (Nome, email, login, e senha de Usuário).
RF02	O sistema deve permitir que o usuário adicione um Objeto que está sendo monitorado informando o identificador e senha cadastrada para o mesmo.
RF03	O sistema deve mostrar em um mapa a localização de um Objeto monitorado.
RF04	O sistema deve mostrar o histórico de localização do Objeto rastreado, quando solicitado pelo usuário.
RF05	O sistema deve permitir o cadastro de um Objeto para ser rastreado, armazenando as seguintes informações: Tipo de objeto que será rastreado (veículo, pessoa, encomenda, outro.), nome e senha.
RF06	O sistema deve permitir a autenticação de um objeto para ser rastreado através do seu identificador e senha cadastrados.
RF07	O sistema deve capturar as coordenadas de localização do Objeto rastreado através do GPS do dispositivo.
RF08	O sistema deve enviar as coordenadas de localização obtida para um *Web Service*.

RF09	O sistema deve capturar as coordenadas de localização do objeto, mesmo que a aplicação seja finalizada no dispositivo rastreado.
RF10	O sistema deve permitir que o usuário ative um alerta, que o notifica sobre a aproximação de determinado objeto.
RF11	O sistema deve disparar uma notificação quando o objeto estiver se aproximando do usuário.

Fonte: Elaborado pelos autores

3.2.2. Arquitetura Cliente-Servidor

A aplicação desenvolvida é baseada na arquitetura cliente-servidor, a qual pode ser definida como uma arquitetura computacional em que vários clientes podem acessar um servidor ao mesmo tempo, em busca de informações. No caso do sistema em questão, o cliente é o aplicativo Android, que faz requisições a um servidor web que, por sua vez, interage com uma base de dados para armazenar ou obter informações solicitadas pelos clientes. Este tipo de arquitetura foi utilizado para o sistema em questão, pelo fato da mesma ter uma maior estabilidade, um maior tráfego de rede para distribuir a localização para diversos usuários, assim como uma maior segurança em controlar o acesso de usuário.

A Figura 13 representa tal modelo arquitetural.

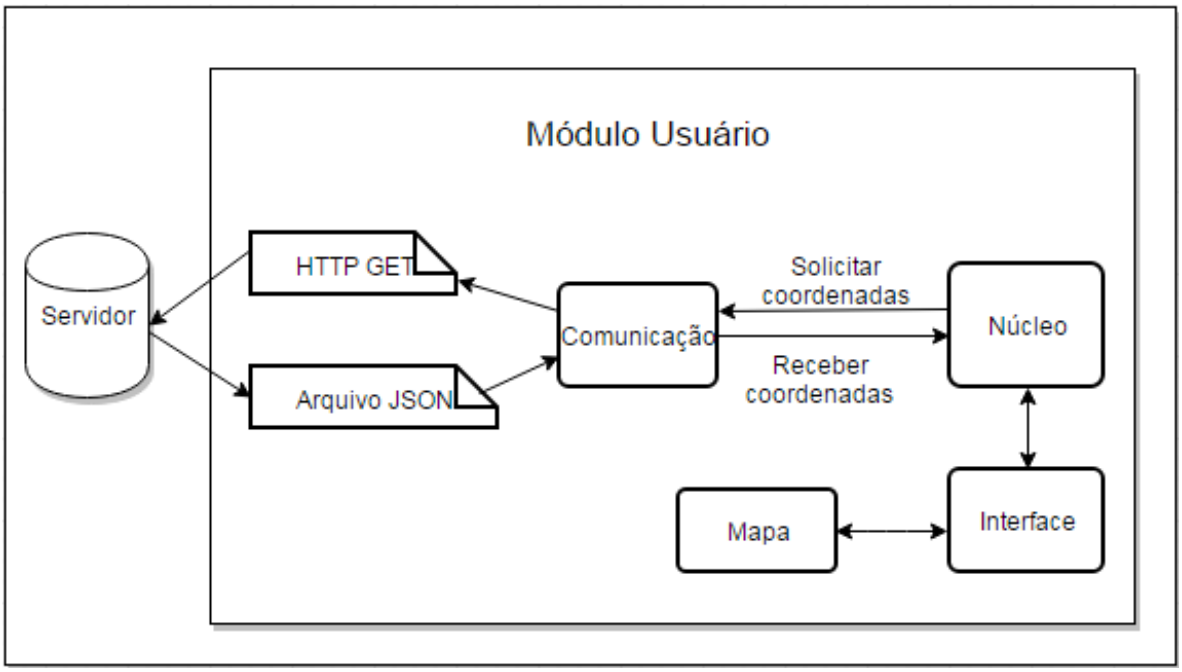
Figura 13 - Arquitetura Cliente-Servidor

Fonte: Elaborado pelos autores

A parte cliente do sistema foi estruturada em submódulos que possuem responsabilidades especificas para o funcionamento da aplicação. A Figura 14 representa a arquitetura do Módulo Usuário e a Figura 15 do Módulo Rastreamento. Como mencionado, anteriormente apesar da divisão do aplicativo em partes distintas, este é um aplicativo único, portanto alguns dos submódulos são compartilhados por ambas as partes. Abaixo segue a descrição de cada um deles;

- **Interface:** responsável pela interação do usuário com os componentes de interface da aplicação. Trata dos dados fornecidos pelo usuário, além de exibir dados recebidos do servidor.
- **Comunicação:** tem por objetivo promover a comunicação do aplicativo com o servidor. Para tal possui funcionalidades que definem a maneira pela qual as requisições devem ser enviadas e como as respostas destas devem ser tratadas.
- **Núcleo:** este submódulo executa o código central da aplicação, solicitando as funcionalidades dos demais submódulos de acordo com aquilo que este necessita. É também responsável pela comunicação entre os demais submódulos da aplicação.

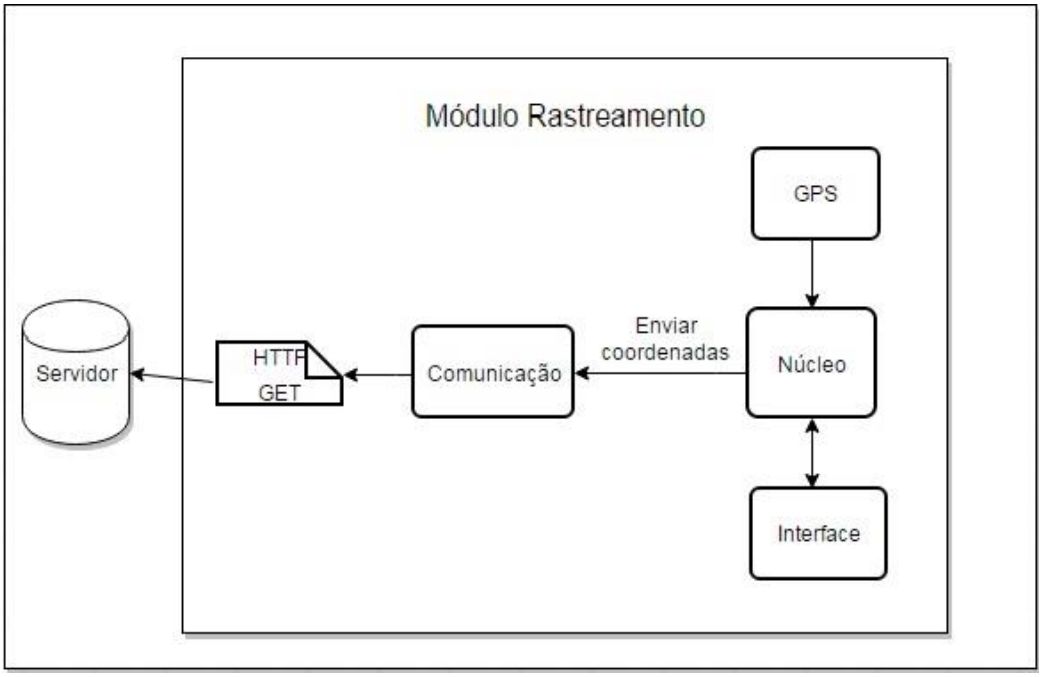
Figura 14 - Arquitetura Cliente/Módulo Usuário



Fonte: Elaborado pelos autores

- **Mapa:** este submódulo é formado por uma única classe, que tem por função tratar os eventos relacionados à visualização da posição do objeto em mapa.

Figura 15 - Arquitetura Cliente/Módulo Rastreamento



Fonte: Elaborado pelos autores

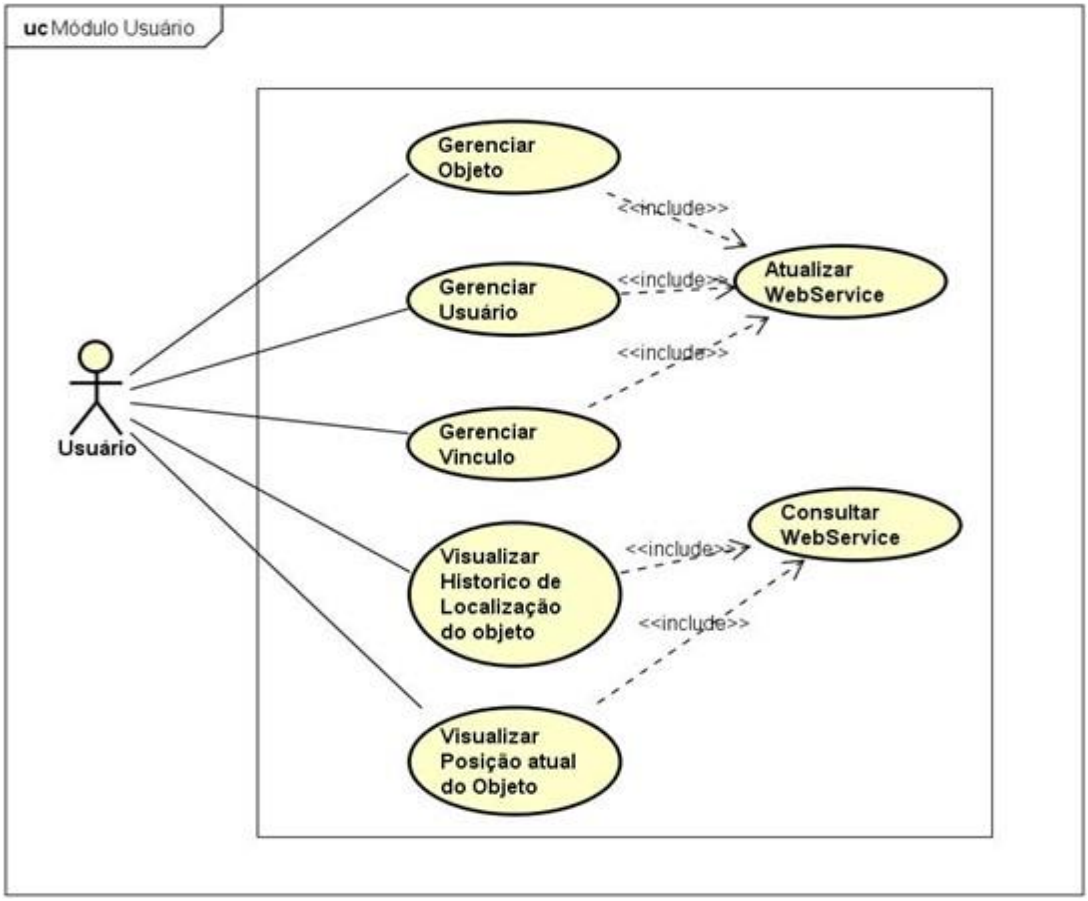
- **GPS:** responsável por obter as informações de localização do dispositivo rastreado, que serão posteriormente enviadas ao servidor.

3.2.3. Casos de Uso

A UML padroniza a documentação das funcionalidades de sistemas por meio do diagrama UC (Casos de Uso). Segundo Furtado e Costa Junior (2010), os casos de uso especificam as funcionalidades de um sistema, segundo diferentes perspectivas dos usuários.

O sistema teve seus casos de uso agrupados segundo os dois módulos existentes. O Diagrama 1 mostra as funcionalidades relacionadas ao módulo Usuário.

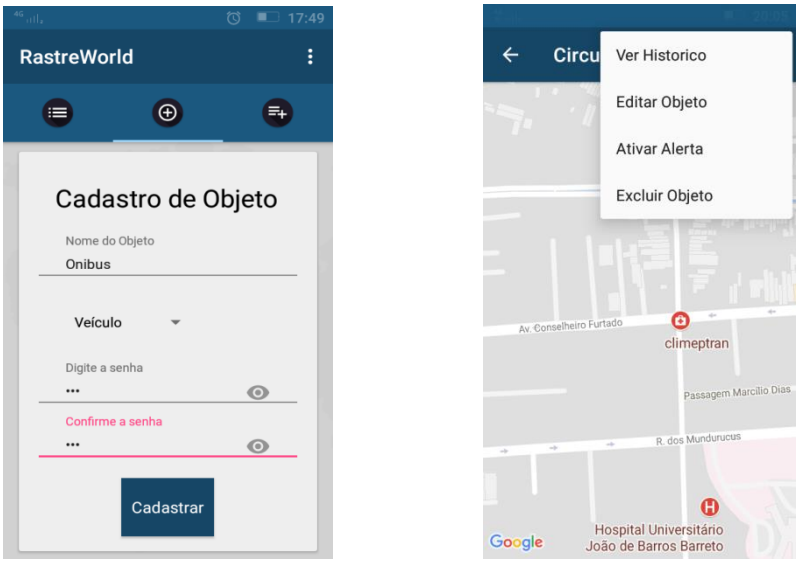
Diagrama 1 - Casos de Uso Módulo Usuário



Fonte: Elaborado pelos autores

Gerenciar objeto: o caso de uso, Gerenciar objeto descreve como o ator usuário interage com o aplicativo para cadastrar, excluir ou alterar um objeto. A Figura 16 ilustra a interface do aplicativo com a qual o usuário irá interagir.

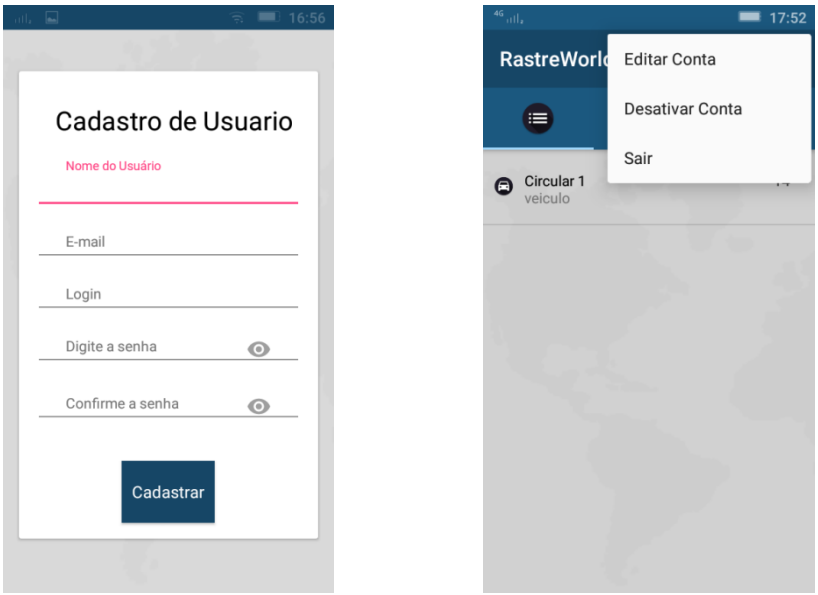
Figura 16 - Ilustração do caso de uso Gerenciar Objeto



Fonte: Elaborado pelos autores

Gerenciar usuário: este caso de uso descreve como o ator usuário interage com o aplicativo para efetuar cadastro, alteração ou desativação de sua conta no sistema. A Figura 17 ilustra a interface do aplicativo com a qual o usuário irá interagir.

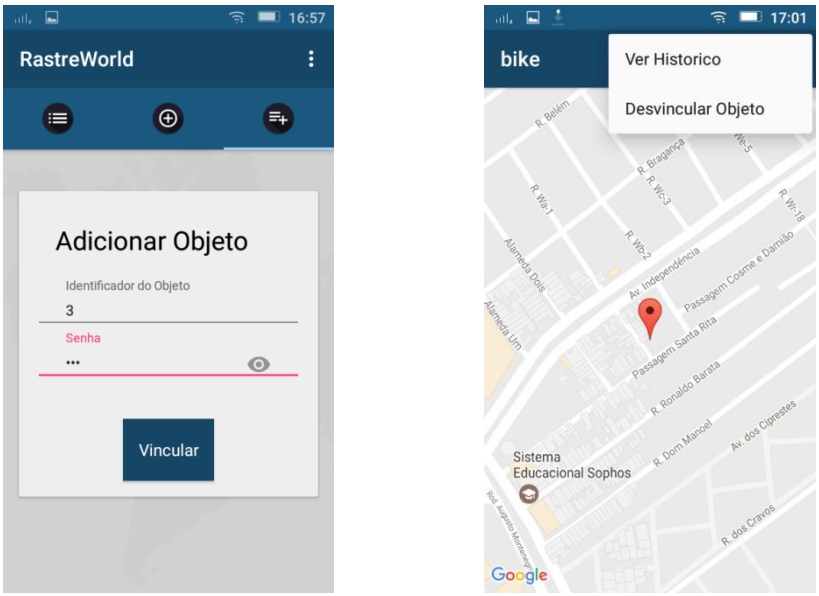
Figura 17 - Ilustração do caso de uso Gerenciar Usuário



Fonte: Elaborado pelos autores

Gerenciar vínculo: o caso de uso Gerenciar Vínculo demonstra como o ator usuário interage com o aplicativo para que este possa vincular ou desvincular um objeto pertencente a outro usuário. A Figura 18 ilustra a interface do aplicativo com a qual o usuário irá interagir.

Figura 18 - Ilustração caso de uso Gerenciar Vinculo



Fonte: Elaborado pelos autores

Visualizar Histórico de localização do Objeto: neste caso de uso o ator usuário interage com o aplicativo para visualizar as posições anteriores do objeto em um mapa. A Figura 19 ilustra a interface do aplicativo com a qual o usuário irá interagir.

Figura 19 - Ilustração do caso de uso Visualizar Histórico

Fonte: Elaborado pelos autores

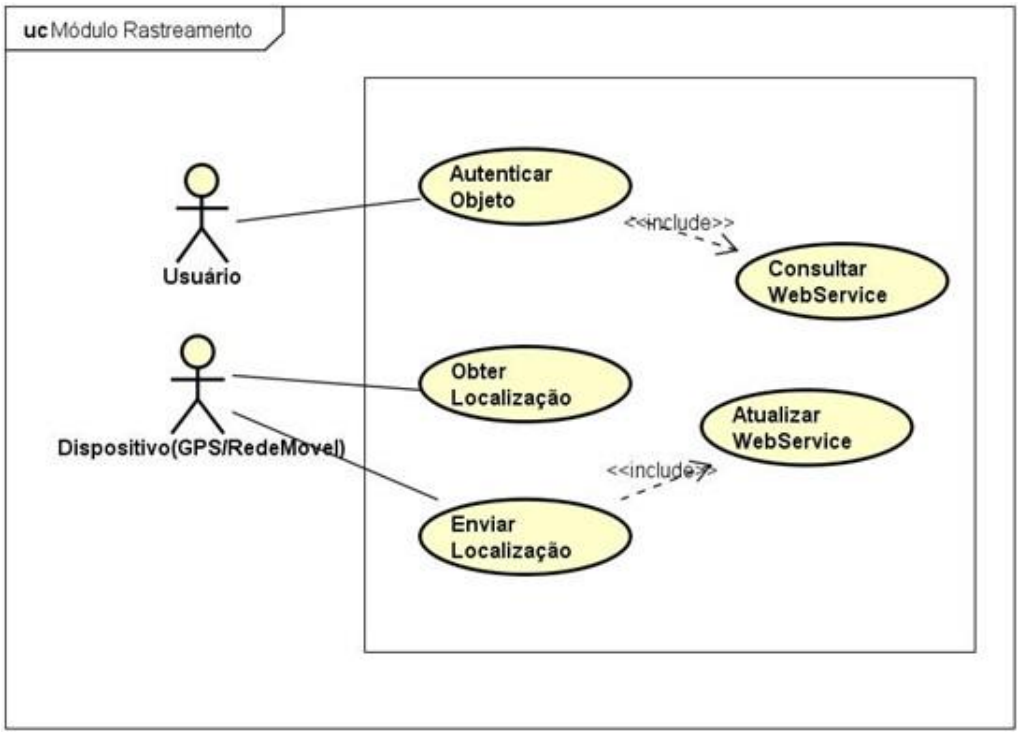
Visualizar Posição Atual do Objeto: neste caso de uso o ator usuário interage com o aplicativo para visualizar a posição atual do objeto em um mapa. A Figura 20 ilustra a interface do aplicativo com a qual o usuário irá interagir.

Figura 20 - Ilustração do caso de uso Visualizar Posição Atual

Fonte: Elaborado pelos autores

O diagrama 2, demonstra as funcionalidades do módulo de rastreamento. Neste o ator usuário é o responsável pelo caso de uso Autenticar Objeto, o qual permite que um dispositivo seja rastreado. Enquanto que o ator DispositivoGPS/RedeMóvel se encarrega dos casos de uso Obter e Enviar localização de um objeto.

Diagrama 2 - Casos de Uso Modulo Rastreamento



Fonte: Elaborado pelos autores

Autenticar objeto: este caso de uso descreve como o ator usuário interage com o aplicativo para realizar a autenticação de um objeto que será rastreado. A Figura 21 ilustra a interface do aplicativo com a qual o usuário irá interagir.

Figura 21 - Ilustração do caso de uso Autenticar Objeto

Fonte: Elaborado pelos autores

Obter localização: este caso de uso descreve como o ator Dispositivo (GPS/RedeMovel), realiza a obtenção da posição do objeto que está sendo rastreado.

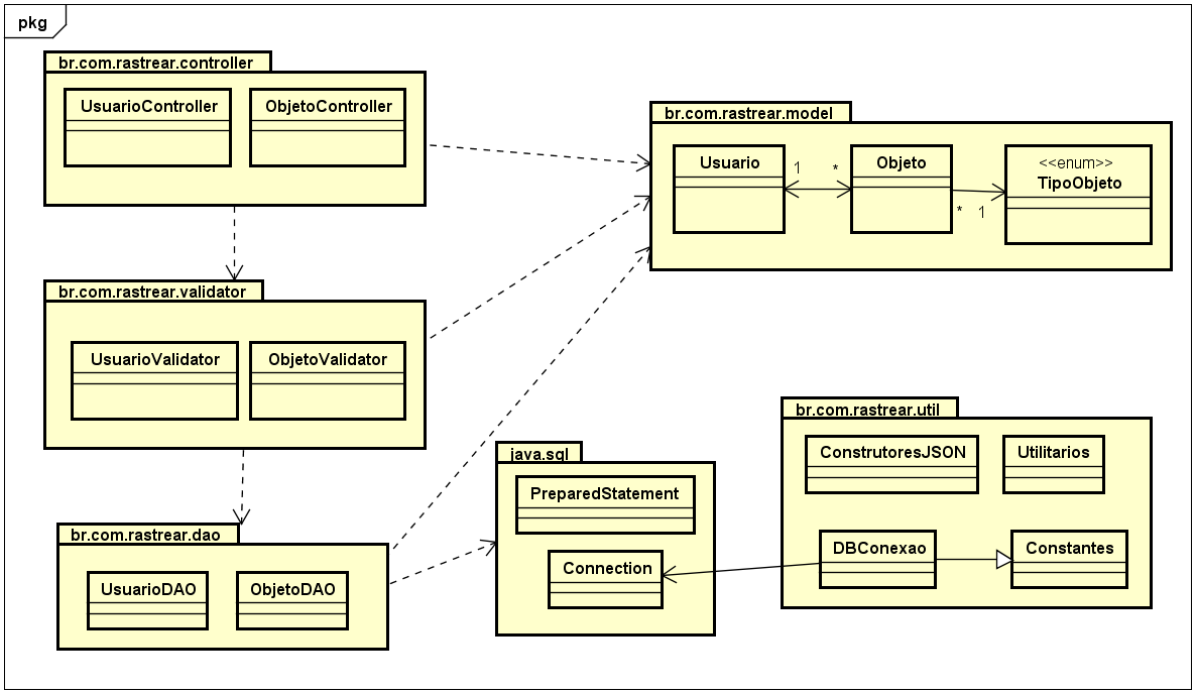
Enviar Localização: este caso de uso descreve como o ator Dispositivo (GPS/RedeMovel), realiza o envio da localização obtida do objeto para o *Web Service*.

3.2.4. Diagrama de Classes

Para definição das entidades de um sistema e seus relacionamentos, a UML propõe o uso do Diagrama de Classes. De acordo com Furtado; Costa Junior (2010), o Diagrama de Classes representa a estrutura estática das classes de um sistema, as quais podem se relacionar com outras, por meio de associações, dependências, herança ou por meio de pacotes de classes.

O Diagrama 3 mostra a estrutura geral de classes do *Web Service*, que está divido em pacotes sendo destacado os quatro mais importantes: pacote br.com.rastrear.model, pacote br.com.rastrear.controller, o pacote br.com.rastrear.validator e o pacote br.com.rastrear.dao.

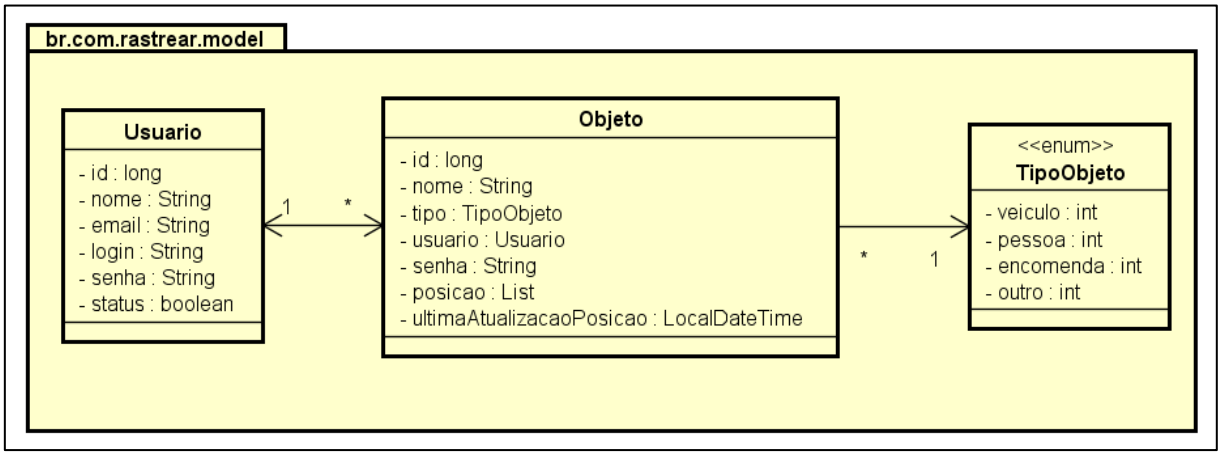
Diagrama 3-Diagrama de Classes *Web Service*



Fonte: Elaborado pelos autores

O pacote br.com.rastrear.model estabelece as entidades a serem tratadas pelo serviço como objetos, visando facilitar a persistência e acesso a dados. O Diagrama 4 mostra essa camada detalhando as classes da mesma.

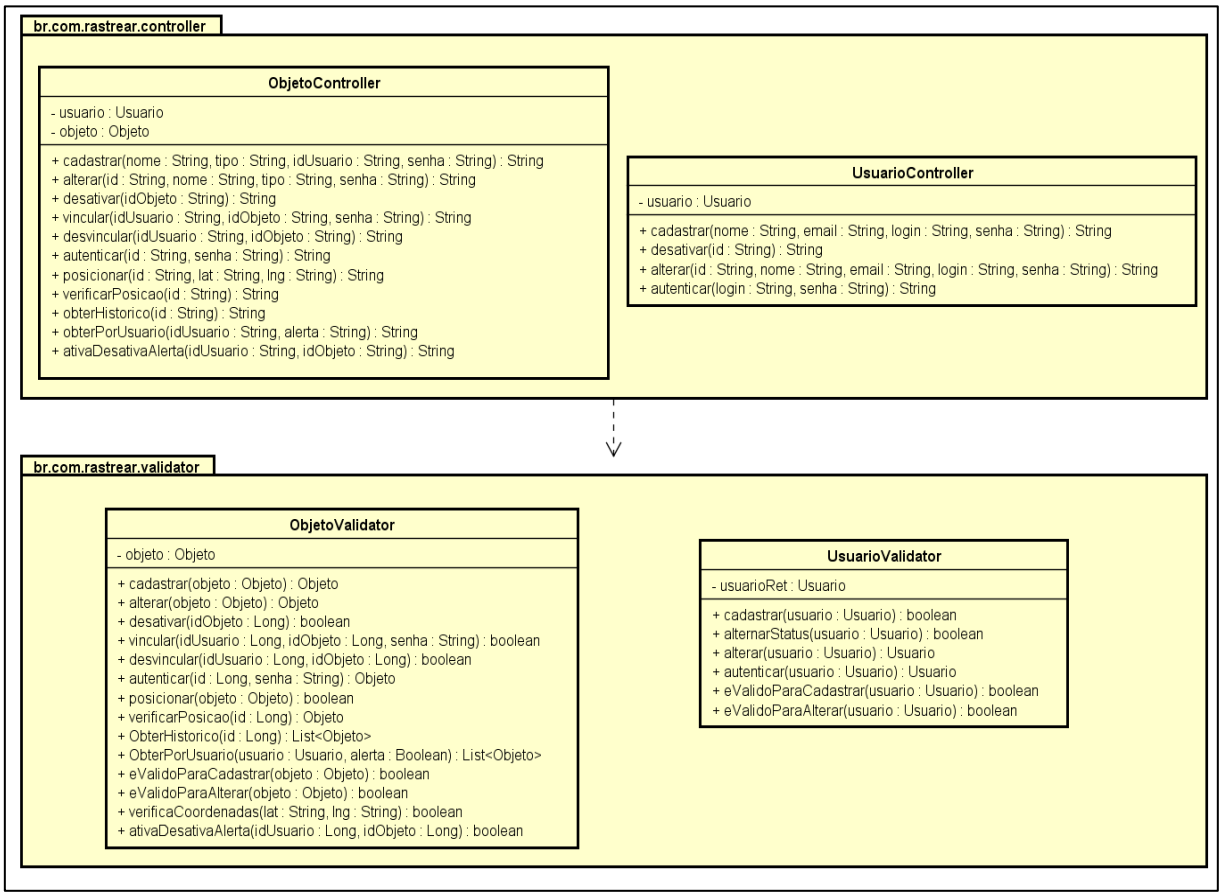
Diagrama 4 - Pacote br.com.rastrear.model



Fonte: Elaborado pelos autores

O pacote br.com.rastrear.controller é encarregado de receber as chamadas dos clientes e seus parâmetros, e repassá-las aos validadores correspondentes. O Diagrama 5 mostra os detalhes do pacote em questão e do pacote br.com.rastrear.validator.

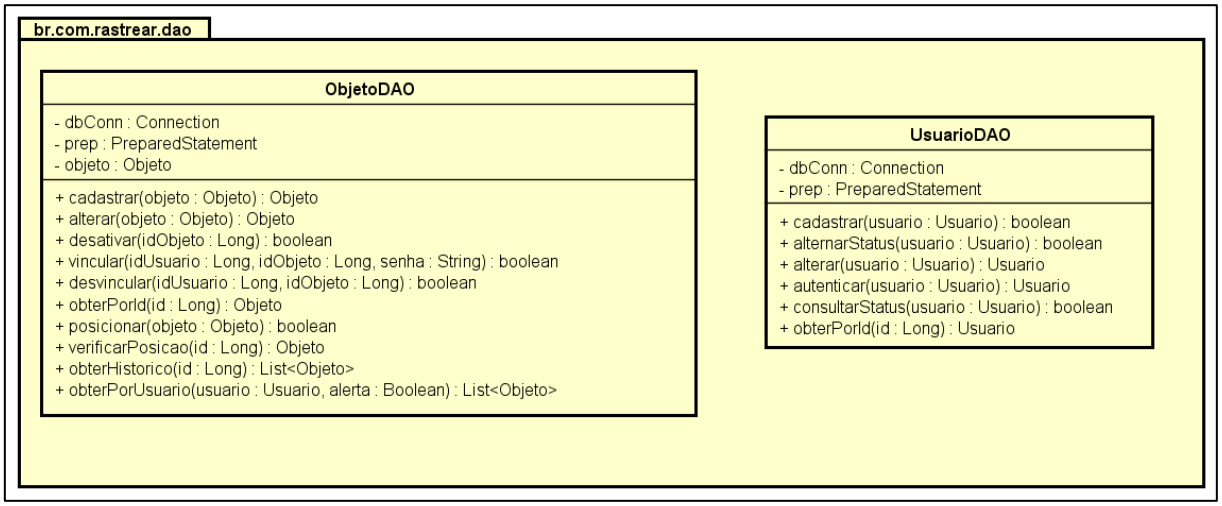
Diagrama 5 - Pacote br.com.rastrear.controller e br.com.rastrear.validator



Fonte: Elaborado pelos autores

O pacote br.com.rastrear.dao trata requisições de consulta e manipulação de dados, sendo a único que possui relação direta com a base. No Diagrama 6 é possível visualizar os detalhes desta camada.

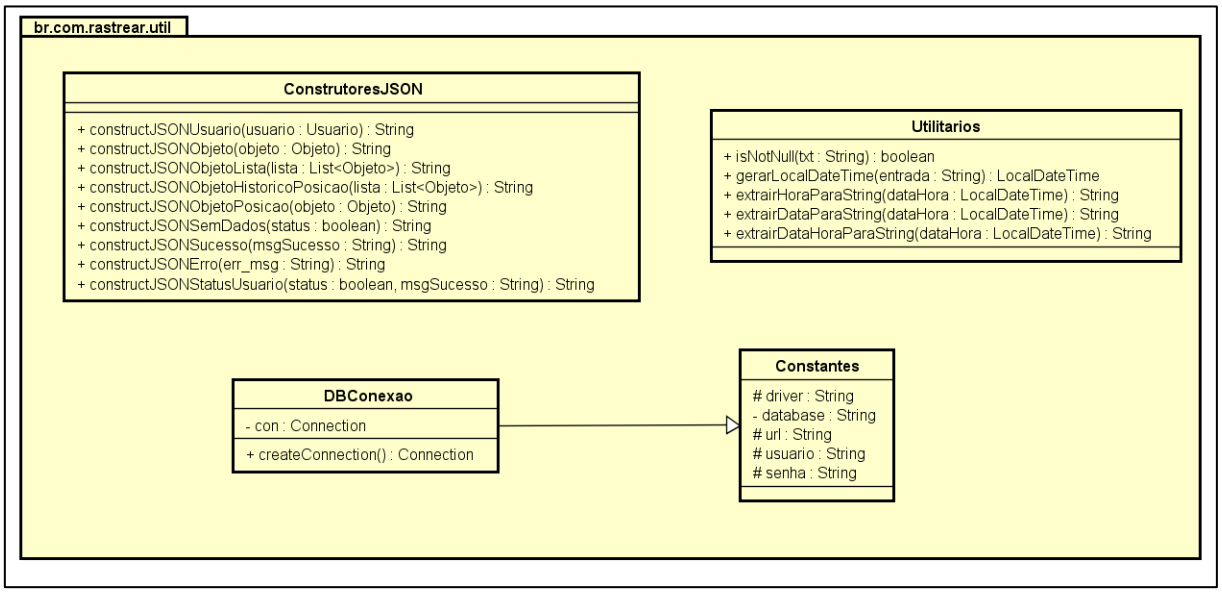
Diagrama 6 - Pacote br.com.rastrear.dao



Fonte: Elaborado pelos autores

Além desses pacotes, tem-se também o pacote br.com.rastrear.util, que abriga classes que são utilizadas pelas classes dos demais pacotes, como a classe *DBConexão* que é utilizada para efetuar uma conexão com a base de dados, e a classe *ConstrutoresJSON*, responsável por codificar as respostas do *Web Service* no formato JSON. O Diagrama 7 detalha este pacote.

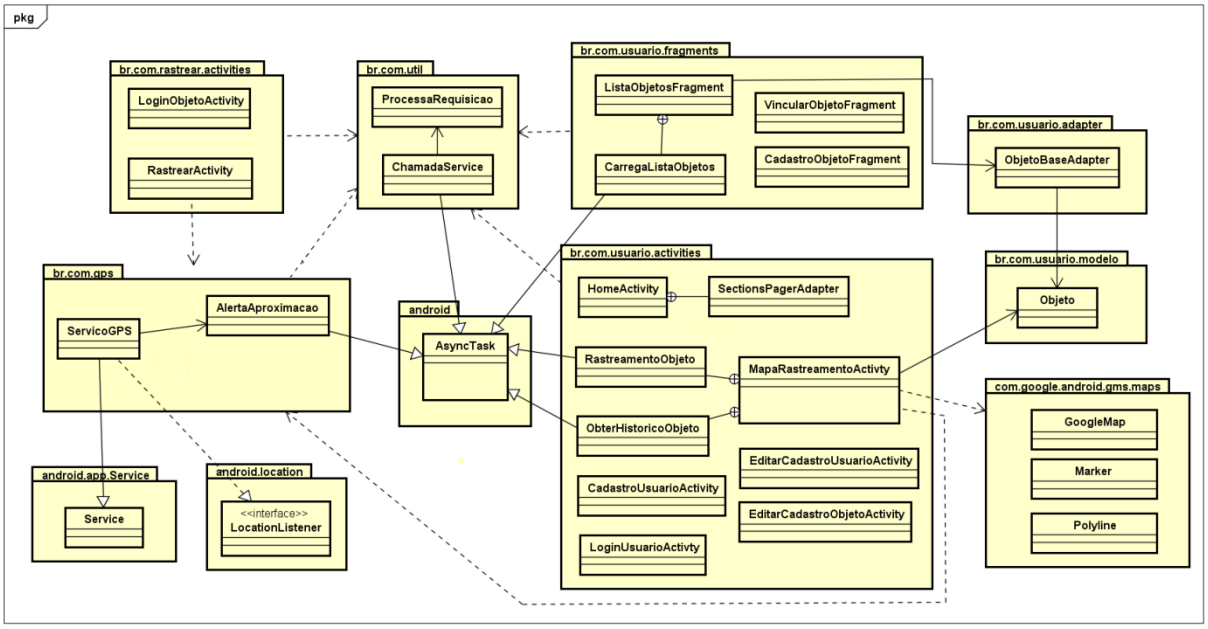
Diagrama 7 - Diagrama Pacote br.com.rastrear.util



Fonte: Elaborado pelos autores

O Diagrama 8 representa a estrutura de classes do aplicativo móvel, onde este é organizado em pacotes que contém as classes de cada módulo do aplicativo, os quais serão detalhados nos próximos Diagramas.

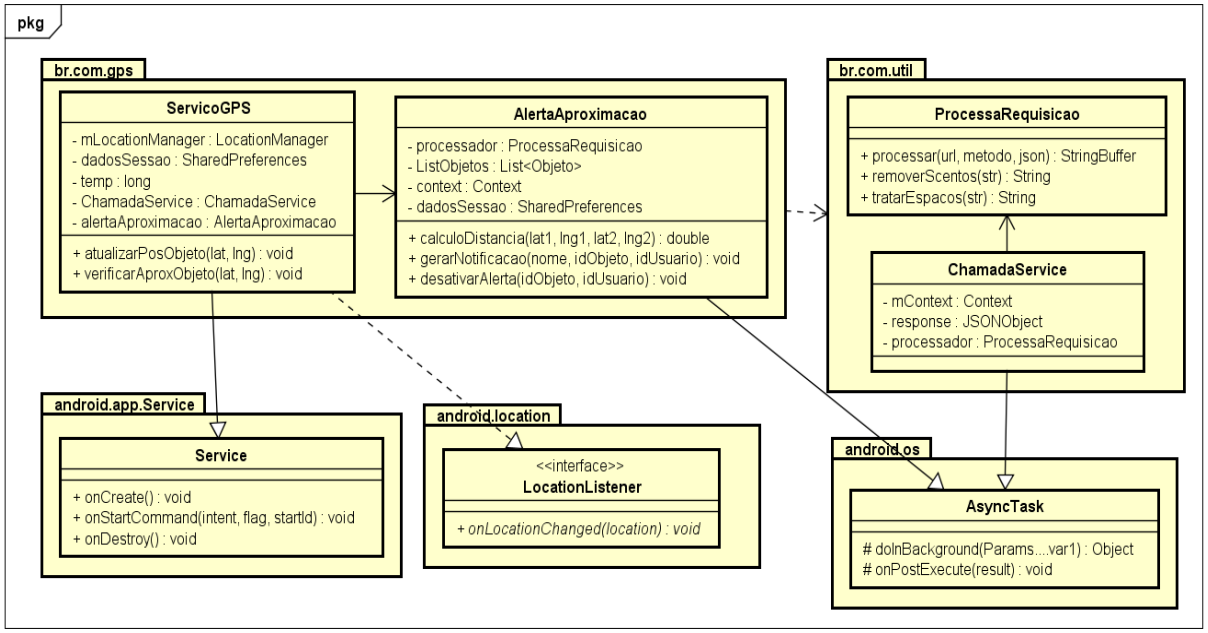
Diagrama 8 - Diagrama de Classes Aplicativo



Fonte: Elaborado pelos autores

O Diagrama 9 representa os pacotes que contém classes que são utilizadas pelos dois módulos do aplicativo. Neste diagrama está o pacote **br.com.gps**, possui a classe responsável pela obtenção das informações de localização do dispositivo através da utilização do GPS do mesmo. Sendo que esta localização será do objeto no caso da utilização do módulo rastreamento ou do usuário no caso de utilização do módulo usuário. E o pacote **br.com.util** que possui as classes *ChamadaService* e *ProcessaRequisição*, as quais são responsáveis por estabelecer conexão e comunicação com o *Web Service*. Estão representados também pacotes nativos do Android, como o **android.app.Service, android.location** e o **android.os**.

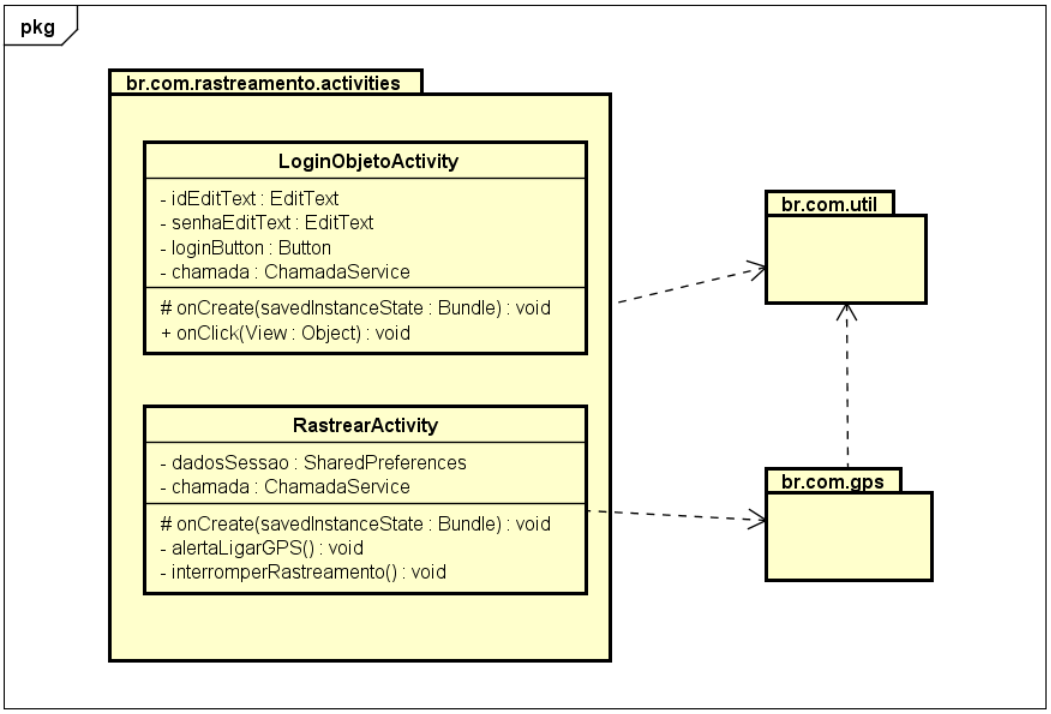
Diagrama 9 - Diagrama de classes utilizados pelos dois módulos do Aplicativo



Fonte: Elaborado pelos autores

O Diagrama 10 contém o pacote **br.com.rastreamento.activities** que possui as classes que representam os componentes de interface do aplicativo do módulo rastreamento.

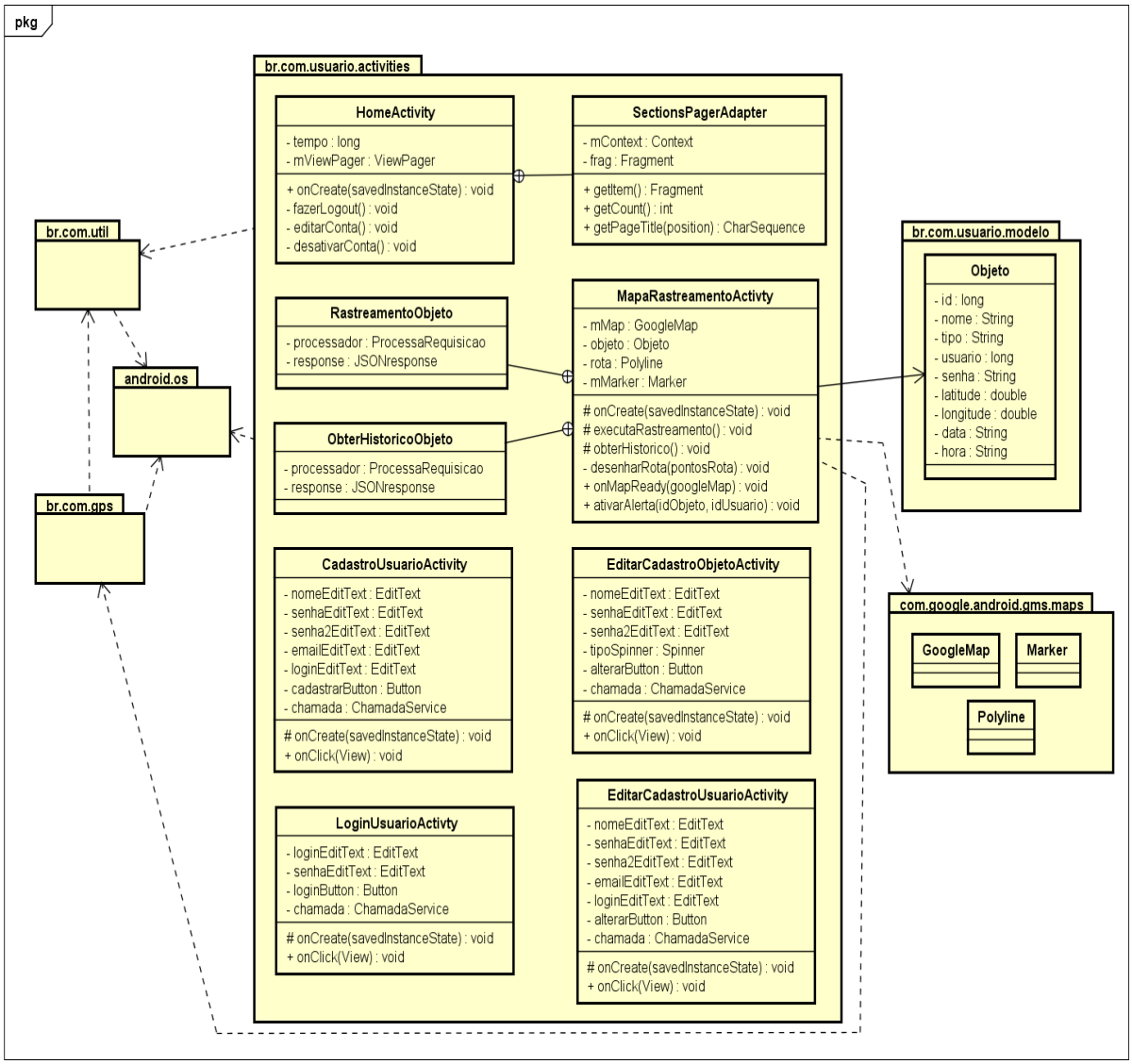
Diagrama 10 - Diagrama de Classes Módulo Rastreamento



Fonte: Elaborado pelos autores

O Diagrama 11 representa os pacotes de classes do módulo usuário. Nele está o pacote **br.com.usuario.activities**, que contém as classes que representam os componentes da interface do aplicativo, do módulo usuário, bem como o pacote **br.com.usuario.modelo**, que possui a classe que recebe os valores da base de dados.

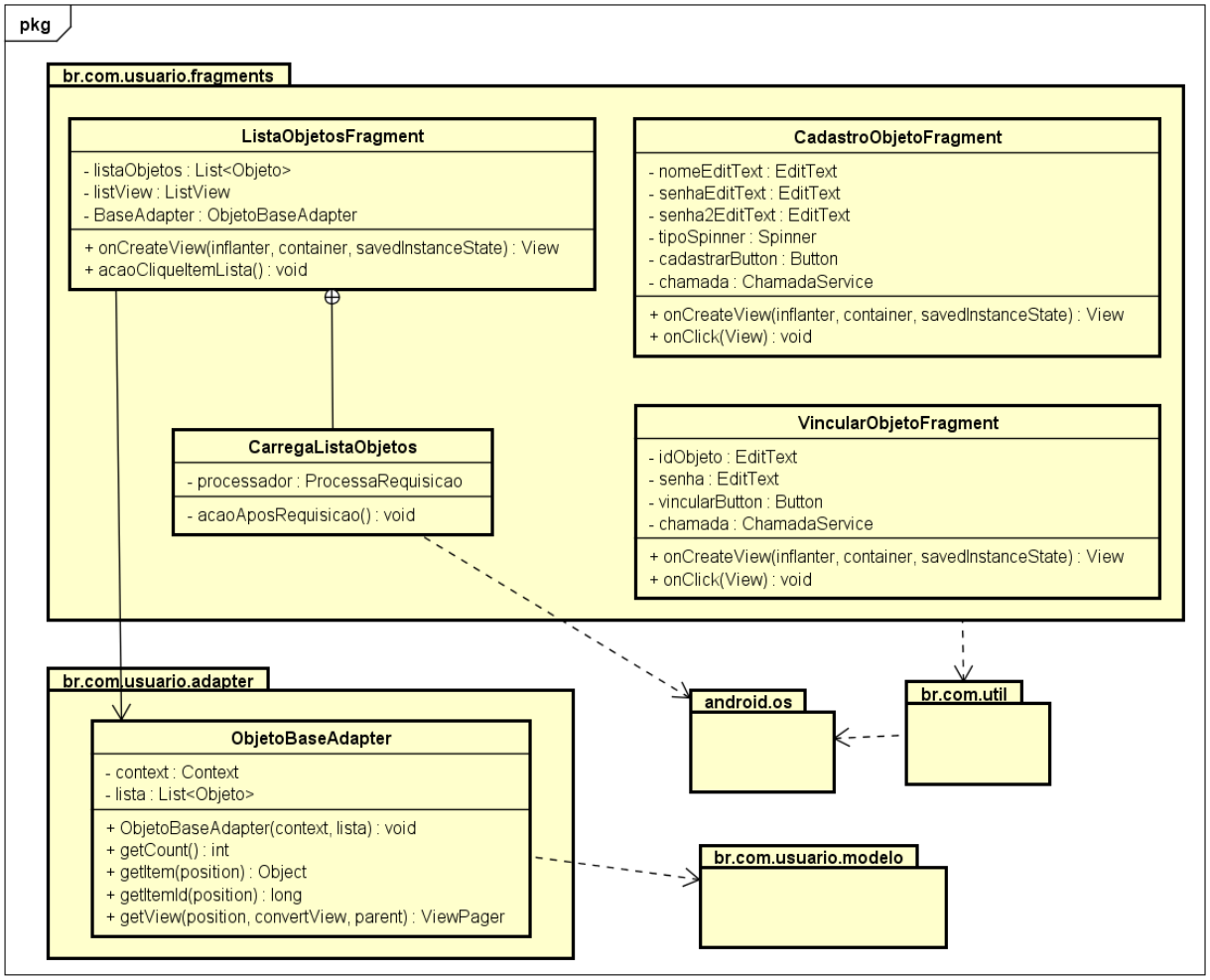
Diagrama 11- Diagrama de Classes Módulo Usuário



Fonte: Elaborado pelos autores

Os pacotes **br.com.usuario.fragments** e **br.com.usuario.adapter** também do módulo usuário, contém classes que representam componentes de interface desse módulo e a classe que tem responsabilidades de controlar o conteúdo de uma lista, respectivamente. O Diagrama 12 detalha esses pacotes.

Diagrama 12 – Diagrama de Classes Módulo Usuário

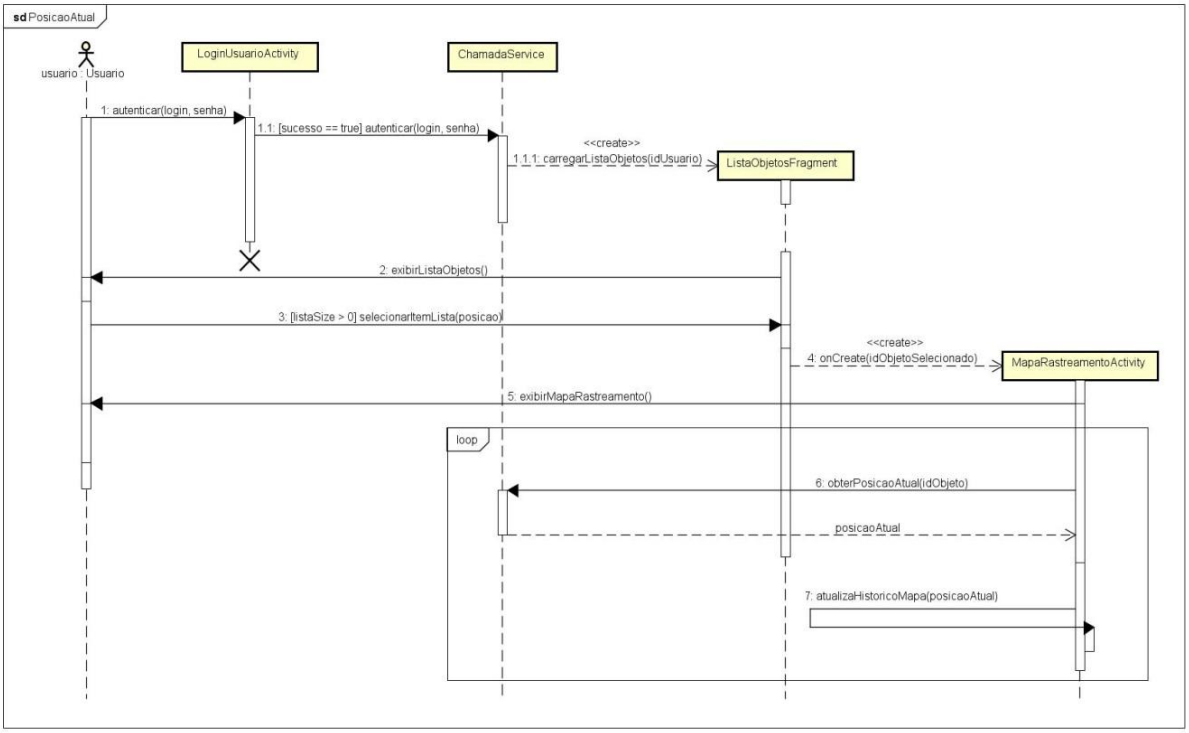


Fonte: Elaborado pelos autores

3.2.5. Diagrama de Sequência

Segundo Furtado e Costa Junior (2010), o diagrama de sequência é utilizado para demonstrar as interações entre os componentes que fazem parte da realização de determinada funcionalidade do sistema. Através dele é possível observar quais componentes participam de uma interação e a sequência das mensagens trocadas entre eles. O Diagrama 13 representa a sequência de interações do caso de uso Visualizar Objeto.

Diagrama 13 - Diagrama de Sequência- Visualizar Objeto



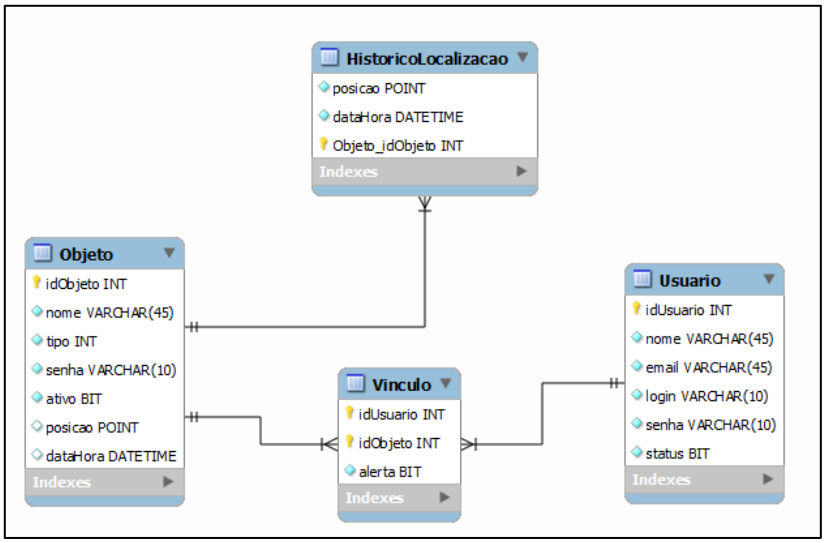
Fonte: Elaborado pelos autores

3.2.6. Banco de Dados

A base de dados utilizada para o armazenamento das informações referente aos objetos e usuários, foi construída utilizando o SGBD MySQL Server versão 5.6. Pois este é um software livre, de fácil manuseio e compatível com várias linguagens de programação (MySQL, 2016).

O Diagrama 6 mostra a estrutura da base de dados do projeto desenvolvido.

Diagrama 14 - Diagrama Entidade Relacionamento



Fonte: Elaborado pelos autores

O modelo utilizado possui uma estrutura simples, sendo composta por apenas 4 entidades. Nos Quadros 2 ao 5 tem-se a descrição das tabelas do banco de dados da aplicação. Os tipos de dados utilizados nos atributos são:

a) *int:* para variáveis numéricas.

b) *varchar*: armazena caracteres alfanuméricos.

c) *datetime:* utilizado para armazenar data e hora.

d) *bit:* para armazenar números definidos 0 ou 1.

c) *point*: para armazenar coordenadas geográficas.

Quadro 2 - Tabela Objeto

Objeto: utilizada para armazenar informações dos objetos rastreados				
Campo	**Descrição**	**Tipo**	**Chave**	**Tamanho**
id_objeto	Identificação única do objeto	Int	PK	
nome	Nome atribuído ao objeto	Varchar		10
tipo	Identifica o tipo do objeto (0-veículo, 1-pessoa, 2- encomenda, 3- outro)	Int		
senha	Senha do objeto	Varchar		10
ativo	Indica se o objeto está ativo ou não	Bit		1
posicao	Armazena as coordenadas geográficas.	Point		
dataHora	Armazena data e hora da ultima atualização	DateTime		

Fonte: Elaborado pelos autores

Quadro 3 - Tabela Usuário

Usuario: utilizada para armazenar informações dos usuários da aplicação				
Campo	**Descrição**	**Tipo**	**Chave**	**Tamanho**
id_usuario	Identificação única do usuario	Int	PK	
nome	Nome completo do usuario	Varchar		20
email	Armazena endereço eletrônico do usuário	Varchar		20
login	Login do usuario	Varchar		10
senha	Senha do usuario	Varchar		10
status	Indica se o usuario está ativo ou não.	Bit		1

Fonte: Elaborado pelos autores

Quadro 4 - Tabela Vinculo

Vinculo: armazena o vínculo entre usuários e objetos cadastrados no sistema.				
Campo	**Descrição**	**Tipo**	**Chave**	**Tamanho**
id_usuario	Identificação única do usuario	Int	FK	

id_objeto	Identificação única do objeto	Int	FK	
alerta	Indica se o objeto está com o alerta de aproximação ativado	Bit		1

Fonte: Elaborado pelos autores

Quadro 5 - Tabela HistoricoLocalizacao

HistoricoLocalização: armazena as informações de posição dos objetos rastreados.				
Campo	**Descrição**	**Tipo**	**Chave**	**Tamanho**
id_objeto	Identificação única do objeto	Int	FK	
posicao	Armazena as coordenadas geográficas.	Point		
dataHora	Armazena data e hora de cada posição	DateTime		

Fonte: Elaborado pelos autores

3.3. Funcionalidades

Esta seção descreve as funcionalidades do aplicativo móvel, por meio do fluxo de interfaces com o usuário. A intenção é disponibilizar todas as funcionalidades de forma intuitiva e direta, além de isolar ambos os módulos. A tela inicial do aplicativo permite a escolha de qual módulo será utilizado pelo dispositivo, conforme mostra a Figura 22.

Figura 22 - Tela Inicial do Aplicativo

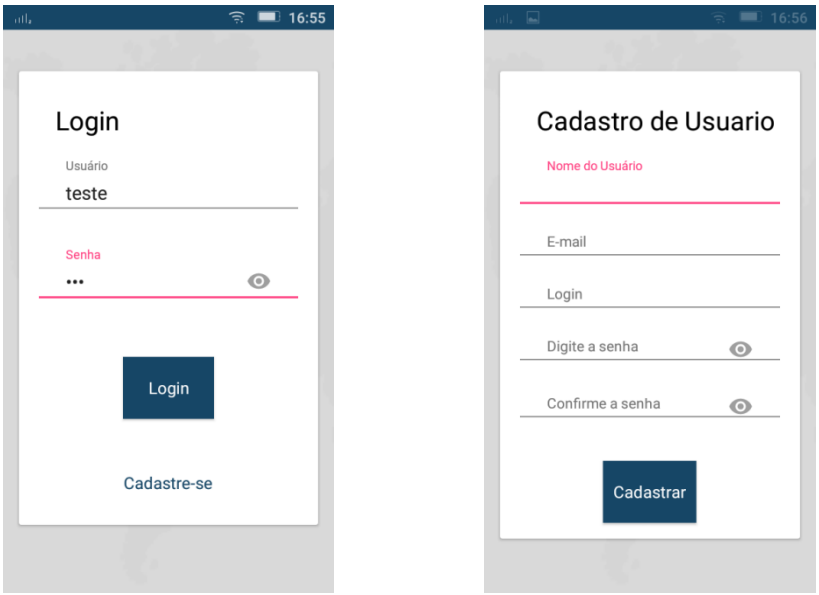
Fonte: Elaborado pelos autores

3.3.1. Telas e Funcionalidades do Módulo Usuário

A opção “Rastrear” permite o acesso ao módulo Usuário, no qual é possível visualizar os objetos rastreados, bem como cadastrar, alterar, excluir um objeto ou adicionar um já

existente. Neste módulo, também é possível cadastrar, alterar e desativar uma conta de usuário. Quando é selecionada a opção “Rastrear”, o usuário é direcionado para tela de login, conforme Figura 23. O usuário deve informar login e senha, para ter acesso às outras funcionalidades do aplicativo.

Figura 23 - Tela de Login e Cadastro de Usuário

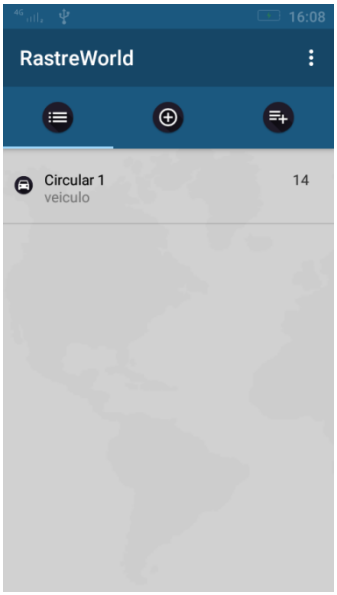


Fonte: Elaborado pelos autores

Caso o mesmo não possua cadastro, basta clicar na opção “Cadastre-se” para obter uma conta. A tela de cadastro é mostrada na Figura 23.

Após autenticação, o usuário será redirecionado para tela principal, que possui três abas: uma com a lista dos objetos que este usuário possui outra para cadastrar um objeto e a última para adicionar um objeto pertencente a outro usuário. A tela principal é mostrada na Figura 24.

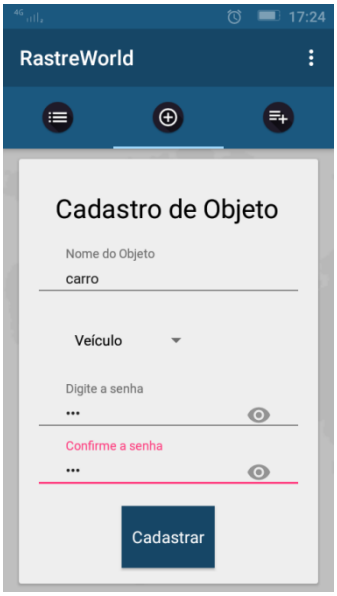
Figura 24 - Tela Principal do Aplicativo



Fonte: Elaborado pelos autores

Cadastro de Objeto: Para cadastrar um objeto, basta escolher a aba de cadastro, localizada na tela principal do aplicativo e informar os dados necessários: nome do objeto, tipo de objeto (pessoa, veículo, encomenda, outros) e senha. Após clicar no botão de cadastro, o aplicativo atualiza a lista de objetos, com o novo objeto cadastro e seu número de identificação. A Figura 25 mostra a tela de cadastro de objeto.

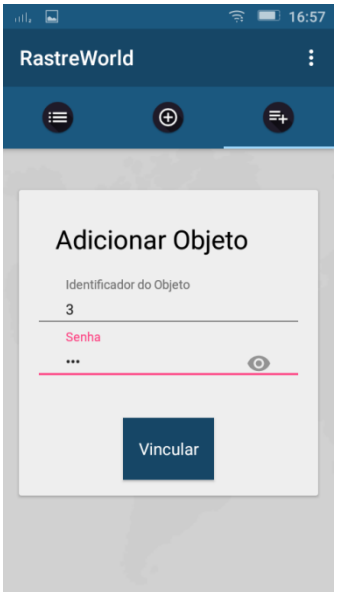
Figura 25 - Tela de Cadastro de Objeto



Fonte: Elaborado pelos autores

Adicionar Objeto: O Aplicativo permite que um usuário adicione a sua lista de objetos um objeto cadastrado por outro usuário, bastando informar o identificador e senha do mesmo. A Figura 26 mostra a tela que vincula um objeto.

Figura 26 - Tela para vincular Objetos



Fonte: Elaborado pelos autores

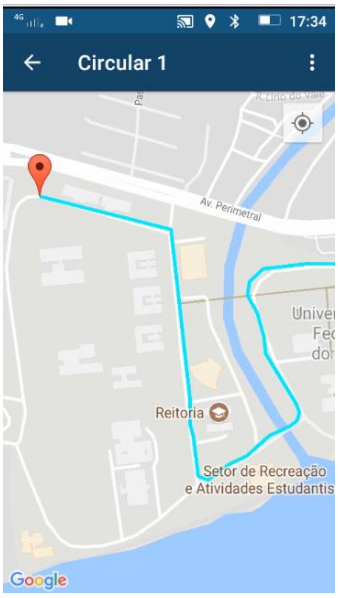
Visualizar Objeto em Mapa: Para que o usuário visualize a posição de um objeto no mapa, basta um clique simples em cima do objeto da lista. Dessa forma, o aplicativo exibe a tela com o mapa e posição atual do objeto, conforme Figura 27.

Figura 27 - Tela de exibição da Posição do objeto

Fonte: Elaborado pelos autores

Histórico de Posições: Com esta opção, é possível visualizar em mapa as posições anteriores de determinado objeto. Esta opção está disponível no menu da tela da Figura 27 que mostra a posição atual do objeto. Ao escolher a opção ver histórico, a tela da Figura 28 é apresentada.

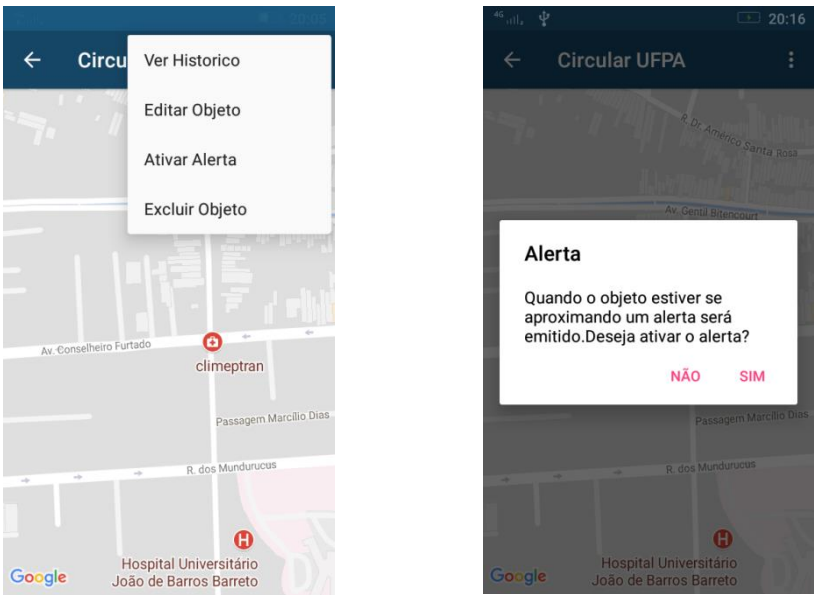
Figura 28 - Tela de exibição do histórico de posições



Fonte: Elaborado pelos autores

Alerta de aproximação: Com esta opção ativada o usuário recebera uma notificação quando o objeto estiver próximo a ele em um raio de 200 metros. Para utilizar esta funcionalidade, o usuário deve escolher a opção “Ativar Alerta” conforma mostra a Figura 29.

Figura 29 - Ativando alerta de aproximação

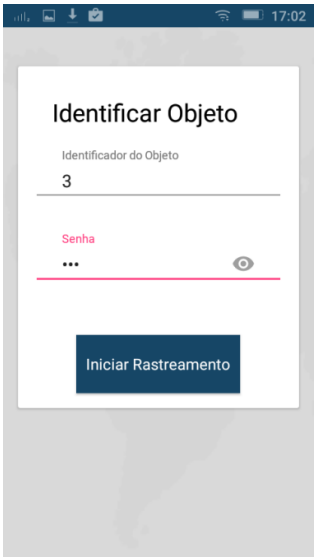


Fonte: Elaborado pelos autores

3.3.2. Telas e Funcionalidades do Módulo Rastreamento

A opção “Ser Rastreado” dará acesso ao módulo rastreamento, no qual, através da autenticação de um objeto, será possível obter as coordenadas de localização do mesmo. Para obter essa localização, o aplicativo acessa o GPS do dispositivo para capturar as coordenadas, que serão enviadas ao *Web Service* para atualização no banco de dados. A Figura 30 mostra a tela de autenticação do objeto a ser rastreado.

Figura 30 - Tela de autenticação de Objeto



Fonte: Elaborado pelos autores

Iniciar Rastreamento: Para iniciar o rastreamento de um objeto, é necessário informar senha e identificador cadastrados para o mesmo. Ao clicar no botão “Iniciar Rastreamento” é realizada a autenticação do objeto junto ao servidor e, então, inicia-se o acesso ao GPS para a obtenção das coordenadas. A Figura 31 mostra a tela após a autenticação do objeto.

Figura 31 - Tela Principal Módulo Rastreamento

Fonte: Elaborado pelos autores

Interromper rastreamento: Para interromper o rastreamento, é necessário clicar no botão Interromper e informar a senha do objeto, conforme Figura 32. Caso a senha esteja correta, o rastreamento é interrompido. Caso contrário, o usuário é notificado de que a senha está incorreta.

Figura 32 - Tela Interromper Rastreamento

Fonte: Elaborado pelos autores

4 IMPLEMENTAÇÃO

Esta seção trata dos aspectos relacionados à implementação do *Web Service* e do aplicativo móvel, demonstrando quais as ferramentas e tecnologias foram utilizadas.

4.1. Web Service

O *Web Service* foi construído baseado na abordagem REST. Dessa maneira todas as informações necessárias para o processamento de uma requisição devem esta contidas na mesma. O tipo de formato de dados escolhido para que as informações trafeguem pela rede, foi o JSON, pois este é um formato mais compacto que XML, e possui suporte nativo no Android.

As principais tecnologias e ferramentas utilizadas foram:

- Linguagem: Java 8
- IDE: Eclipse Luna
- Servidor de aplicação: Apache Tomcat 8
- Bibliotecas externas:
  - MySQL Connector
  - Jersey
- Formato de transferência de dados: JSON

O *Web Service* deste trabalho disponibiliza os recursos e sua manipulação de maneira que atendam a cada regra de negócio do projeto. Esses recursos estão disponíveis em uma camada de controle que são acessíveis via requisição HTTP.

Na Listagem 1 tem-se trecho da classe *UsuarioController* que representa o recurso usuário, tal representação é identificada pela anotação *@Path ("/usuario")*. Dentro desta classe, estão todos os métodos que permitem a manipulação deste recurso. O método *cadastrar* recebe uma anotação *@POST*, que indica que este método será acionado através de uma requisição HTTP POST. A anotação *@Consumes* e *@Produces* indicam respectivamente, o formato em que os dados podem ser consumidos pelo método e o formato que a resposta deste pode ser gerada. Neste caso, o método irá consumir e gerar resposta no formato JSON.

Listagem 1 – Trecho de Código Classe *UsuarioController*

```
@Path("/usuario")
public class UsuarioController {

//http://host:porta/RastrearWS/usuario

@POST
```

```
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public String cadastrar(JSONObject obj){

                    [...]

 }

//http://host:porta/RastrearWS/usuario?login=loginusuario&senha=senhausuario

@GET
@Produces(MediaType.APPLICATION_JSON)
public String autenticar(@QueryParam("login") String login,
                         @QueryParam("senha") String senha){
                        [...]      }

[...] }
```

Fonte: Elaborado pelos autores

Estas anotações são fornecidas pela biblioteca Jersey, e tem por finalidade facilitar o mapeamento das URI's que darão acesso ao serviço por parte do cliente. Desta maneira, para utilizar o serviço de cadastro de usuário, a URI utilizada será /usuario e a URL completa *http://host:porta/RastrearWS/usuario*, sendo que os dados referentes ao novo usuário serão enviados em formato JSON no corpo da requisição, bem como o método HTTP utilizado, neste caso, o método POST.

É possível também enviar parâmetros extras na própria url para utilizar um serviço. Conforme se observa no método autenticar na Listagem 1, o qual utiliza o método HTTP GET para recuperar o usuário correspondente ao login e senha enviada na url. No Quadro 6, estão todas as URI's definidas para os serviços deste projeto.

Quadro 6- URI's para acesso aos serviços do Web Service

URI	Método	Dados extras	Efeito
/usuario	Post	Formato JSON	Cadastra um novo usuário
	Get	?login=x&senha=y	Recupera um usuário (autenticação).
/usuario/{id}	Put	Formato JSON	Altera dados de um usuário.
	Delete	-	Exclui um usuário.
/usuario/{id}/objeto	Post	Formato JSON	Usuário cadastra um novo objeto.
	Get	?alerta=true/false	Recupera a lista de objetos de um usuário
/objeto/{id}	Get	?senha=x	Recupera um objeto (autenticação)
	Put	Formato JSON	Altera um objeto
	Delete	-	Exclui um objeto
/objeto/{id}/posicao	Get	-	Recupera a posição de um objeto
	Put	Formato JSON	Atualiza a posição de um objeto

/objeto/{id}/historico	Get	-	Recupera o histórico de um objeto
/objeto/{id}/alerta	Put	?idusuario=x	Ativa alerta de aproximação de um objeto
/objeto/{id_obj}/ {id_usr}/vincular	Post	-	Vincula um objeto a um usuário.
	Delete	-	Desvincula um usuário de um objeto vinculado.

Fonte: Elaborado pelos autores

A camada de controle repassa os parâmetros recebidos a uma camada inferior encarregada de validar estes dados. A esta camada se deu o nome de validador e seu objetivo é assegurar a integridade da informação a ser enviada ou recebida da base de dados. A Listagem 2, exemplifica um método da camada de validação referente ao cadastro de usuários. O método auxiliar *eValidoParaCadastrar* verifica se os dados de cadastro estão nulos. Se os dados forem válidos a camada de banco de dados é acionada, caso contrário mensagens de erro são lançadas como resposta ao controlador.

Listagem 2 - Código Método *cadastrar* (Camada Validator)

```
public Boolean cadastrar(Usuario usuario) throws Exception {
        if (eValidoParaCadastrar(usuario)) {
                try {
                        return dao.cadastrar(usuario);
                } catch(SQLException e){
                        String msgErro = e.getErrorCode() == 1062 ? "Login já
existe!":"Ocorreu um erro no cadastro.";
                        throw new Exception(msgErro);
                } catch (Exception e) {
                        throw new Exception("Erro ao cadastrar usuário.");
                }
        }
        else
           throw new Exception("Dados de cadastro insuficientes.");}

private boolean eValidoParaCadastrar(Usuario usuario){
        return Utilitarios.isNotNull(usuario.getNome()) &&
                Utilitarios.isNotNull(usuario.getLogin()) &&
                Utilitarios.isNotNull(usuario.getSenha()) &&
                Utilitarios.isNotNull(usuario.getEmail());}

public class Utilitarios {
        public static boolean isNotNull(String txt) {
             return txt != null && txt.trim().length() > 0 }
}
```

Fonte: Elaborado pelos autores

A manipulação propriamente dita da base de dados se dá por meio da Camada de Acesso aos Dados, identificada por DAO. A abertura e fechamento de conexão com a base de dados são gerenciados por meio de uma classe utilitária que faz uso da interface padrão de

conexão JDBC, disponível no Java EE. A Listagem 3 demonstra o método que cadastra novos usuários na base de dados. Ele abre uma nova conexão com a base MySQL, realiza a transação de inserção dos dados, em seguida encerrando a conexão. O retorno para camada de validação é um valor booleano indicando se houve sucesso na operação.

Listagem 3 - Código Método *cadastrar* (Camada DAO)

```
public boolean cadastrar(Usuario usuario) throws SQLException, Exception {
      try {
            try {
                  dbConn = DBConexao.createConnection();
               } catch (Exception e) {
                  e.printStackTrace();
            }
            String query = "INSERT into usuario (nome, email, login, senha) values
('"+usuario.getNome()+ "','" + usuario.getEmail() + "','"+ usuario.getLogin() + "','" +
usuario.getSenha() + "')";
            prep = dbConn.prepareStatement(query);
            int records = prep.executeUpdate(query);
            return records > 0;
      } catch (SQLException sqle) {
            throw sqle;
      } catch (Exception e) {
            throw e;
      } finally {
            if (dbConn != null) dbConn.close();
      }
   }
```

Fonte: Elaborado pelos autores

Como forma de dar subsídios às operações com JSON foi criada a classe *ConstrutoresJSON*, contendo métodos estáticos com o objetivo de construir as respostas às requisições do aplicativo. A Listagem 4 exemplifica a construção de uma lista de objetos JSON ao qual comporá uma lista de objetos a ser exibida no aplicativo.

Listagem 4 - Código Método *constructJSONObjetoLista*

```
public static String constructJSONObjetoLista(List<Objeto> lista) throws Exception {
                  JSONArray array = new JSONArray();
                  for(Objeto objeto : lista){
                        JSONObject obj = new JSONObject();
                        try {
                              obj.put("id", objeto.getId());
                              obj.put("nome", objeto.getNome());
                              obj.put("tipo", objeto.getTipo());
                              obj.put("idUsuario", objeto.getUsuario().getId());
                              [...]
                              array.put(obj);
                        } catch (JSONException e) {
                              throw new Exception("Erro ao gerar o JSON.");
                        }
            }
```

```
        return array.toString();
    }
```

Fonte: Elaborado pelos autores

A Figura 33 demonstra um exemplo de resposta no formato JSON.

Figura 33 - Resposta do Web Service em Formato JSON

Fonte: Elaborado pelos autores

4.2. Aplicativo Móvel

O aplicativo móvel foi desenvolvido para o sistema operacional Android. O ambiente de desenvolvimento utilizado foi o Android Studio, que é a ferramenta oficial para desenvolvimento de aplicativos Android, o qual utiliza a linguagem de programação Java. Para obtenção das informações de localização do dispositivo fez-se o uso da tecnologia GPS, bem como a API Google Maps para a visualização em mapa das posições do mesmo.

A implementação do aplicativo levou em consideração o fato do mesmo possuir dois módulos distintos, como já mencionados anteriormente. Desta maneira, foi criada uma *activity* que permite a escolha, por parte do usuário, de qual módulo será utilizado no dispositivo em que o mesmo encontra-se instalado. A Listagem 5 mostra trecho do código da *activity* mencionada.

Listagem 5 - Trecho de Código da classe *InicialActivity*

```
public class InicialActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_inicial);

        LinearLayout loginUsuario = (LinearLayout) findViewById(R.id.inicialRastrearButton);
        LinearLayout loginObjeto = (LinearLayout) findViewById(R.id.inicialRastreamentoButton);
         // método chamado caso a opção seja módulo usuário
        loginUsuario.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent in = new Intent(getApplicationContext(), LoginUsuarioActivity.class);
                in.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                startActivity(in);
                finish();
            }
        });
```

```
        //// método chamado caso a opção seja módulo rastreamento
        loginObjeto.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent in = new Intent(getApplicationContext(), LoginObjetoActivity.class);
                in.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                startActivity(in);
                finish();
            }
        });
       [...]
    }
```

Fonte: Elaborado pelos autores

Para realizar a comunicação do aplicativo com o *Web Service*, foi implementada a classe *ChamadaService*. Esta classe estende a classe *AsyncTask*, do Android, a qual é utilizada para efetuar operações em background e que também possibilita a publicação do resultado das operações na thread principal (*UIThread*) da aplicação. A Listagem 6 mostra um trecho de código desta classe, dando destaque aos métodos sobrescritos *doInBackground* e *onPostExecute*. Eles, respectivamente, iniciam uma requisição REST e tratam o JSON retornado.

Listagem 6 - Trecho do Código da classe *ChamadaService*

```
public class ChamadaService extends AsyncTask<String , Void, JSONObject> {
    //atributos
    [...]
    @Override
    protected JSONObject  doInBackground(String... strings) {
        try {
            String urlServico = strings[0];
             //obtém o método HTTP que será utilizado
            String metodo = dados.get("metodo").toString();
            //obtém o json com dados que serão enviados junto a requisição HTTP
            JSONObject json = (JSONObject) dados.get("jsonObject");
            StringBuffer stringBuffer = processador.processar(urlServico, método,json);
            response = new JSONObject(stringBuffer.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
     [...]
    @Override
    protected void onPostExecute(String string) {
        super.onPostExecute(string);
        try {
            [...]
                if(response.has("msgSucesso"))
                  Toast.makeText(contexto, response.getString("msgSucesso"),
Toast.LENGTH_SHORT).show();
                [...]
```

```
            else {
            if(response.has("erro"))
               Toast.makeText(contexto, response.getString("erro"),
Toast.LENGTH_SHORT).show();
            [...]       }
      catch(JSONException e){
         Toast.makeText(contexto, "Erro ao ler o JSON (" + e.getMessage() + ")",
Toast.LENGTH_SHORT).show();
      }
      progress.dismiss();
   }
}
```

Fonte: Elaborado pelos autores

Outra classe importante neste processo, é a classe *ProcessaRequisicao*. Ela é utilizada pela classe *ChamadaService*, através da chamada do método processar, que recebe como parâmetro uma URL, o método HTTP a ser utilizado e um *JSON* com dados que serão enviados no corpo da requisição. E através de uma conexão HTTP, acessa o *Web Service* para obter ou atualizar informações. A Listagem 7 mostra a classe *ProcessaRequisicao*.

Listagem 7 - Código da classe *ProcessaRequisicao*

```
public class ProcessaRequisicao {
    public StringBuffer processar(String urlServico, String método, JSONObject json) throws
JSONException {
       try {
          urlServico = removerAcentos(urlServico);
          urlServico = trataEspacos(urlServico);
          URL url = new URL(urlServico);

          HttpURLConnection connection;
          connection = (HttpURLConnection) url.openConnection();
          connection.setRequestMethod(metodo); // seta o método que será utilizado

         if (json.length() > 0 ){//
            [....]
            //informa o formato que o dado está sendo enviado.
           connection.setRequestProperty( "Content-Type", "application/json" );
          //informa em que formato deve ser a resposta
          connection.setRequestProperty("Accept", "application/json");

          OutputStream os = connection.getOutputStream();
          OutputStreamWriter osw = new OutputStreamWriter(os, "UTF-8");
          osw.write(json.toString());
          osw.flush();
          osw.close();
          }
          connection.connect();

          InputStream inputStream = connection.getInputStream();
          BufferedReader bufferedReader = new BufferedReader(new
```

```
InputStreamReader(inputStream));
        StringBuffer stringBuffer = new StringBuffer();
        String linha;

        while((linha = bufferedReader.readLine()) != null)
        stringBuffer.append(linha);
        inputStream.close();
        connection.disconnect();
        return stringBuffer;
        } catch (IOException e) {
        e.printStackTrace();
.....}
    throw new JSONException("Erro ao processar a requisição!");
  }
}
```

Fonte: Elaborado pelos autores

Quando uma funcionalidade da aplicação necessita estabelecer uma conexão com o *Web Service* a classe *ChamadaService* é instanciada, sendo repassado o contexto, a URL, o método e o JSON montado com os dados necessários para o processamento da requisição. A Listagem 8, mostra trecho do código do método que solicita o cadastro de um novo usuário.

Listagem 8 - Trecho de Código da classe *CadastroUsuarioActivity*

```
public class CadastroUsuarioActivity extends AppCompatActivity {
    [...]
    loginButton.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
          // json com os dados do novo usuario
          JSONObject json = new JSONObject();
              try {
                    json.put("nome", nomeEditText.getText().toString());
                    json.put("login", loginEditText.getText().toString());
                    json.put("email", emailEditText.getText().toString());
                   json.put("senha",senhaEditText.getText().toString());
              } catch (JSONException e) {
          e.printStackTrace();}
          // URL completa (http://host:porta/RastrearWS/usuario)
          String url = IWS.urlBase+IWS.entidadeUsuario;

          chamada = new ChamadaService(LoginUsuarioActivity.this);
          chamada.addInfo("metodo","POST"); //método HTTP que será utilizado
          chamada.addInfo("jsonObject", json); //json que será enviado no corpo da requisição

          [...]  // passando URL para classe ChamadaService
          chamada.execute(url);
        }
    });
}
```

Fonte: Elaborado pelos autores

Em algumas solicitações será necessário enviar dados extras na própria URL, para tal será utilizado um objeto *RequestParams*, conforme se observa na Listagem 9.

Listagem 9 - Trecho de Código da classe *LoginUsuarioActivity*

```
public class LoginUsuarioActivity extends AppCompatActivity {
    [...]
    loginButton.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
          // parâmetros para montar a URL
          RequestParams params = new RequestParams();
          params.put("login",loginEditText.getText().toString());
          params.put("senha",senhaEditText.getText().toString());
          // URL completa (http://host:porta/RastrearWS/usuario?login=x&senha=y
          String url = IWS.urlBase+IWS.entidadeUsuario+"?"+params;
          chamada = new ChamadaService(LoginUsuarioActivity.this);
          chamada.addInfo("metodo","GET");//método HTTP que será utilizado
          [...]  // passando URL para classe ChamadaService
          chamada.execute(url);
        }
    });
}
```

Fonte: Elaborado pelos autores

Para obtenção dos dados de localização do dispositivo, foi criada a classe *ServicoGPS*, a qual estende a classe *Service* do Android, que é um componente que possibilita executar tarefas de longa duração em *background* e que não possui interface gráfica (MONTEIRO, 2012). Dessa maneira, mesmo que o aplicativo seja fechado (retirado da memória) ou minimizado, as coordenadas continuaram a ser capturadas. Essa classe pode ser utilizada pelos dois módulos do aplicativo. O módulo rastreamento, utiliza esta classe para obter as coordenadas do dispositivo (objeto que está sendo rastreado) e envia-las ao *Web Service* para serem armazenadas na base de dados. Por outro lado, o módulo usuário faz o uso desta classe para obter as coordenadas do dispositivo (localização do usuário), e utiliza tal informação para verificar a distância que determinado objeto encontra-se deste, sendo disparada uma notificação caso o objeto esteja próximo.

O Android disponibiliza a classe *LocationManager*, responsável por fornecer acesso aos serviços de localização. Através do método *requestLocationUpdates* obtém-se as coordenadas de localização do dispositivo. A classe *ServicoGPS* implementa a interface *LocationListener*, utilizada para receber as atualizações de localização. A Listagem 10 mostra trecho do código da classe em questão. No método *onStartCommand* a classe *LocationManager é* recuperada através do método *getSystemService(Context.LOCATION_SERVICE)*, já o método *requestLocationUpdates* é

chamado passando como parâmetros o provedor de localização, o intervalo de tempo e a distância entre as atualizações de localização.

Listagem 10 - Trecho de Código da classe *ServicoGPS*

```
public class ServicoGPS extends Service implements LocationListener  {
    private LocationManager lm;
    private Double lat;
    private Double lng;
    private String id;

   @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //recupera o LocationManager
        lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
       //obtém as coordenadas de localização
        lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 4000, 0,this);
        return START_STICKY;
    }
     [. . .]
```

Fonte: Elaborado pelos autores

Após a obtenção das coordenadas, é invocado o método *onLocationChanged* da classe *LocationListener*, onde a latitude e a longitude obtidas são recuperadas e enviadas ao Web Service através do método *atualizarPosicaoObjeto*, caso seja o módulo rastreamento que esteja fazendo uso da mesma, ou será chamado o método *verificarAproximaçãoObjeto* no caso do uso pelo módulo usuário conforme mostra a Listagem 11.

Listagem 11 - Código Método *onLocationChanged()*

```
@Override
public void onLocationChanged(final Location location) {
   Double lat = location.getLatitude();
   Double lng = location.getLongitude();

 if(dadosSessao.contains("idObjeto")){
      atualizarPosicaoObjeto(lat, lng);
   }else{
      verificarAproximacaoObjeto(lat, lng);
   }
}
```

Fonte: Elaborado pelos autores

No método *atualizarPosicaoObjeto* é montada a URL, que contém as informações de localização do objeto a qual é repassada para a classe *ChamadaService* que se encarregara de enviar estas ao servidor. O método *verificarAproximaçãoObjeto*, também monta uma URL, porém com as informações que permitem obter os objetos que estão com alerta de aproximação ativado. Esta URL, bem como as informações da latitude e longitude obtidas

serão repassadas a classe *AlertaAproximação*, cuja tarefa é verificar e alertar a aproximação dos objetos do usuário. Estes dois métodos estão descritos na listagem 12.

Listagem 12 - Código Método *atualizarPosicaoObjeto()* e *verificarAproximaçãoObjeto()*

```
public void atualizarPosicaoObjeto(final Double latitude, final Double longitude){
   String id = dadosSessao.getString("idObjeto", "");
   JSONObject json = new JSONObject();
    try {
            json.put("lat",latitude.toString() );
            json.put("lng",longitude.toString() );
        } catch (JSONException e) {
        e.printStackTrace();
   }
   //URL Completa (http://host:porta/RastrearWS/objeto/id/posicao)
   String url = IWS.urlBase + IWS.entidadeObjeto + id +IWS.servicoPosicionar;

   ChamadaService chamadaService = new ChamadaService(getApplicationContext());
   chamadaService.addInfo("metodo","PUT");
   chamadaService.addInfo("jsonObject",json);
   chamadaService.execute(url);
}

public void verificarAproximacaoObjeto(Double latitude,Double longitude){
   String idUsersessao = dadosSessao.getString("idUsuario","");
   RequestParams params = new RequestParams();
   params.put("alerta","true");
    //URL Completa (http://host:porta/RastrearWS/usuario/id/objeto?alerta=true)
   String url = IWS.urlBase +
IWS.entidadeUsuario+idUsersessao+IWS.entidadeObjeto+"?"+params;

   alertaAproximacao = new AlertaAproximacao(getApplicationContext());
   alertaAproximacao.addInfo("idUsersessao", idUsersessao);
   alertaAproximacao.addInfo("latitude",latitude.toString()); //latitude do usuario
   alertaAproximacao.addInfo("longitude",longitude.toString());//longitude do usuario
   alertaAproximacao.execute(url);
}
```

Fonte: Elaborado pelos autores

A classe *AlertaAproximação* estende a classe *AsyncTask*, dessa maneira no seu método *doInBackground* é processada a requisição que obtém as informações de posição dos objetos que estão com o alerta ativado. Após obter o resultado no método *onPostExecute* é realizada a verificação da distância entre o objeto e o usuário, através do método *calculoDistancia*, se a distância entre eles for menor ou igual a duzentos metros, é chamado o método *criarNotificação* o qual é responsável por disparar uma notificação na barra de status do dispositivo. Trecho do código dessa classe pode ser visualizado na Listagem 13.

Listagem 13 – Trecho código Classe *AlertaAproximacao*

```
public class AlertaAproximacao extends AsyncTask <String, Void, JSONArray> {
  //atributos
  [...]
   @Override
   protected  JSONArray doInBackground(String... strings) {
      String urlServico = strings[0];
     String metodo = "GET"; //metodo http que será utilizado
     [......]
      try {
           StringBuffer stringBuffer = processador.processar(urlServico, metodo, json);
          JSONArray array = new JSONArray(stringBuffer.toString());

         for (int i = 0; i < array.length(); i++) {
            Long id = array.getJSONObject(i).getLong("id");
            String nome = array.getJSONObject(i).getString("nome");
            Double lat = array.getJSONObject(i).getDouble("lat");
            Double lng = array.getJSONObject(i).getDouble("lng");
            Long idUsuario = array.getJSONObject(i).getLong("idUsuario");

            Objeto objeto = new Objeto(id, nome, lat, lng, idUsuario);
            listaObjetos.add(objeto);
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
      return null;
   }

   @Override
   protected void onPostExecute(JSONArray jsonArray) {
      super.onPostExecute(jsonArray);
      if (listaObjetos.size() > 0) {
         Double latUsuario = Double.parseDouble(dados.get("latitude").toString());
         Double lngUsuario = Double.parseDouble(dados.get("longitude").toString());
         String idUsersessao = dados.get("idUsersessao").toString();
         for (Objeto o : listaObjetos) {
            Double latObjeto = o.getLat();
            Double lngObjeto = o.getLng();
            Double resultado = calculoDistancia(latUsuario, lngUsuario, latObjeto, lngObjeto);

               if (Math.round(resultado) <= 200) {

               criarNotificacao(o.getNome(), o.getId().toString(), o.getUsuario().toString());
               desativarAlerta(o.getId().toString(), idUsersessao);
            }
         }
      } else {
         contexto.stopService(new Intent(contexto, ServicoGPS.class));
      }
  [...]
   }
```

Fonte: Elaborado pelos autores

A Listagem 14, mostra como é realizado o cálculo que obtém a distância entre o objeto e o usuário. Este método irá receber como parâmetro as coordenadas de localização do usuário e do objeto. O método calcula a distância geográfica entre os dois pontos, a qual leva em consideração a curvatura da Terra.

Listagem 14 – Código método *calculoDistancia*()

```
private Double calculoDistancia(Double lat1, Double lng1, Double lat2, Double lng2) {
    double raiodaterra = 6371;//raio da terra kilometros
    double diferencaLat = Math.toRadians(lat2 - lat1);
    double diferencaLng = Math.toRadians(lng2 - lng1);
    double sindLat = Math.sin(diferencaLat / 2);
    double sindLng = Math.sin(diferencaLng / 2);
    double a = Math.pow(sindLat, 2) + Math.pow(sindLng, 2)
          * Math.cos(Math.toRadians(lat1))
          * Math.cos(Math.toRadians(lat2));
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double distancia = raiodaterra * c;

    return distancia * 1000; //em metros
}
```

Fonte: adaptado de https://thiagovespa.com.br/blog/2010/09/10 (2010)

A Listagem 15 mostra o método responsável por cria às notificações que alertam o usuario da aproximação do objeto rastreado. Este método recebe como parâmetro o nome e o id do objeto e o id do usuário da sessão, esses parâmetros são repassados como dados extras na *Intent*, componente utilizado para iniciar uma nova instância de uma activity, neste caso a activity *MapaRastreamentoActivity*. Sendo esta *Intent* repassada como parâmetro para uma *PendingIntent*, a qual permite que outro componente execute a ação definida anteriormente na *Intent*, no caso quando o usuário seleciona a notificação é chamada a *PendingIntent*. Para definir as características da notificação é utilizada a classe *NotificationCompat.Builder*. É possível definir o ícone, título e texto da notificação, bem como vibração e toque padrão do dispositivo. Em seguida este é repassado a uma instância da classe *Notification*, sendo esta repassada para a classe *NotificationManager* a qual ira colocar a notificação na barra de status.

Listagem 15 – Código método *criarNotificação()*

```
private void criarNotificacao(String Nome, String idObjeto, String idUsuario) {

    Intent i = new Intent(contexto.getApplicationContext(), MapaRastreamentoActivity.class);
    i.putExtra("id", idObjeto);
    i.putExtra("nome", Nome);
    i.putExtra("idUsuario", idUsuario);
    i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TASK);
```

```
        PendingIntent p = PendingIntent.getActivity(contexto, 0, i,
PendingIntent.FLAG_UPDATE_CURRENT);

        Uri som = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);

        NotificationCompat.Builder nb = new NotificationCompat.Builder(contexto)
                .setContentTitle("Alerta")
                .setContentText(Nome + " está próximo!")
                .setSound(som)
                .setVibrate(new long[]{150, 300, 150, 600})
                .setAutoCancel(true)
                .setSmallIcon(R.mipmap.marker)
                .setContentIntent(p)
                .setLargeIcon(BitmapFactory.decodeResource(contexto.getResources(),
R.mipmap.marker));

        NotificationManager notificationManager = (NotificationManager)
contexto.getSystemService(Context.NOTIFICATION_SERVICE);
        Notification n = nb.build();
        notificationManager.notify(Integer.parseInt(idObjeto), n);
    }
```

Fonte: Elaborado pelos autores

Para o acompanhamento da movimentação dos objetos, foi utilizada uma *activity* do tipo *Google Maps Activity*, a qual possui a estrutura básica para utilização de mapas, através da API Google Maps.

De acordo com o site oficial da *Google Developers*, para que seja possível utilizar mapas em um aplicativo, é necessária a obtenção de uma chave de acesso, que pode ser gerada através do site *Google API Console*, de maneira gratuita.

A Listagem 16, mostra o trecho da *activity MapaRastreamentoActivity*, responsável pelo carregamento do mapa com a posição do objeto selecionado.

Listagem 16 - Trecho do Código *MapaRastrementoActivity*

```
public class MapaRastreamentoActivity extends AppCompatActivity implements
OnMapReadyCallback {

    […]
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_mapa_rastreamento);

        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
         // inicia rastreamento do objeto
        executaRastreamento();
```

```
    }
    […]
```

Fonte: Elaborado pelos autores

Em seu método *onCreate* é chamado o método *executaRastreamento*. Este método possui uma *thread* que realiza a chamada ao servidor de tempos em tempos, para obter a posição atualizada do objeto. Este método pode ser observado na Listagem 17.

Listagem 17 - Trecho do código Método *executaRastreament()*

```
protected void executaRastreamento() {
    Thread t = new Thread() {
        public void run() {
            try{
                while (true){
                    sleep(3000);// tempo que realiza chamada ao servidor em milissegundos
                    String id = objeto.getId().toString();
                    //URL completa (http:host:porta/RastrearWS/objeto/id/posicao
                    String url = IWS.urlBase+IWS.entidadeObjeto+id +IWS.servicoRastreamento;
                    new RastreamentoObjetoService().execute(url);
                }
            } catch(InterruptedException e){
                [...]
            }
        }
    };
    t.start();
}
```

Fonte: Elaborado pelos autores

5 ESTUDO DE CASO

Neste capítulo será descrito o estudo de caso realizado dentro da Universidade Federal do Pará. O aplicativo foi utilizado para realizar o rastreamento do ônibus que trafega dentro da universidade, conhecido como Circular. Por meio deste estudo de caso foi possível analisar os benefícios que o uso deste aplicativo pode trazer para a comunidade acadêmica.

5.1. Visão Geral

A Universidade Federal do Pará (UFPA) encontra-se em meio às maiores universidades do Brasil e é a maior instituição da Região Norte. Atualmente é composta por mais de 50 mil pessoas, desta forma, no Campus Belém circulam mais de 18 mil pessoas entre estudantes, professores, técnicos, comunidade em geral, entre outros (UFPA, 2015).

A instituição possui uma área territorial bem extensa, sendo esta, dividida em Campus I-Básico, Campus II-Profissional e Campus III-Saúde. Para facilitar o deslocamento da comunidade acadêmica dentro do campus são disponibilizados ônibus, conhecidos como Circular, que trafegam diariamente, percorrendo uma rota que cobre praticamente toda a extensão da Universidade.

O serviço de transporte é realizado por dois ônibus de grande porte, gratuitamente, dentro da instituição. A rota estipulada pela Direção de Infraestrutura da Prefeitura do Campus possui um total de 6 quilômetros e 200 metros (UFPA, 2010).

5.2. Ambiente de Teste

Para realização do teste foram utilizados três *smartphones* com características diferentes conforme mostra a Tabela 3. Um dos *smartphones* utilizou o módulo rastreamento, este foi colocado dentro do ônibus Circular, para, assim, realizar o rastreamento do mesmo. Nos outros dois *smartphone* foi utilizado o módulo usuário. Para a realização da comunicação dos *smartphones* com o servidor foi utilizada a rede móvel de telefonia (3G/4G).

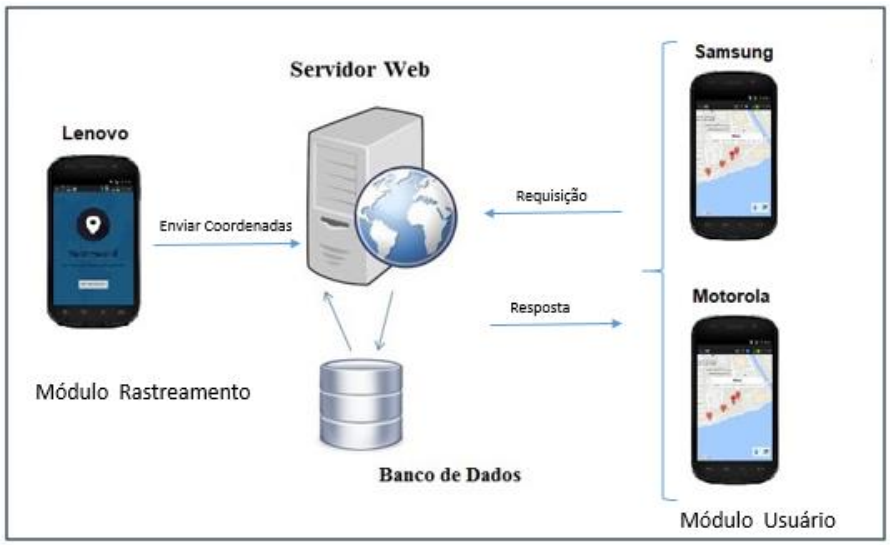
Tabela 3 - Configuração dos smartphones utilizados no teste do aplicativo.

Fabricante	**Modelo**	**Versão do Android**	**Dimensão da Tela**	**Memória**	**Processador**
Samsung	Galaxy Fame	4.1	3,5”	512 MB	Single-Core 1 GHz
Lenovo	Vibe B	6.0	4,5”	1 GB	Quad-Core 1 GHz
Motorola	MotoG4 Plus	7.1	5,5”	2 GB	Octa-Core 1,5 GHz

Fonte: Elaborado pelos autores

Após a instalação do aplicativo, foi realizado o cadastro e autenticação de usuário e objeto/ônibus nos *smartphones* de acordo com o papel definido para cada um deles conforme mostra a Figura 34. Sendo assim, iniciou-se o monitoramento do ônibus por parte do usuário e o alerta de aproximação foi ativado.

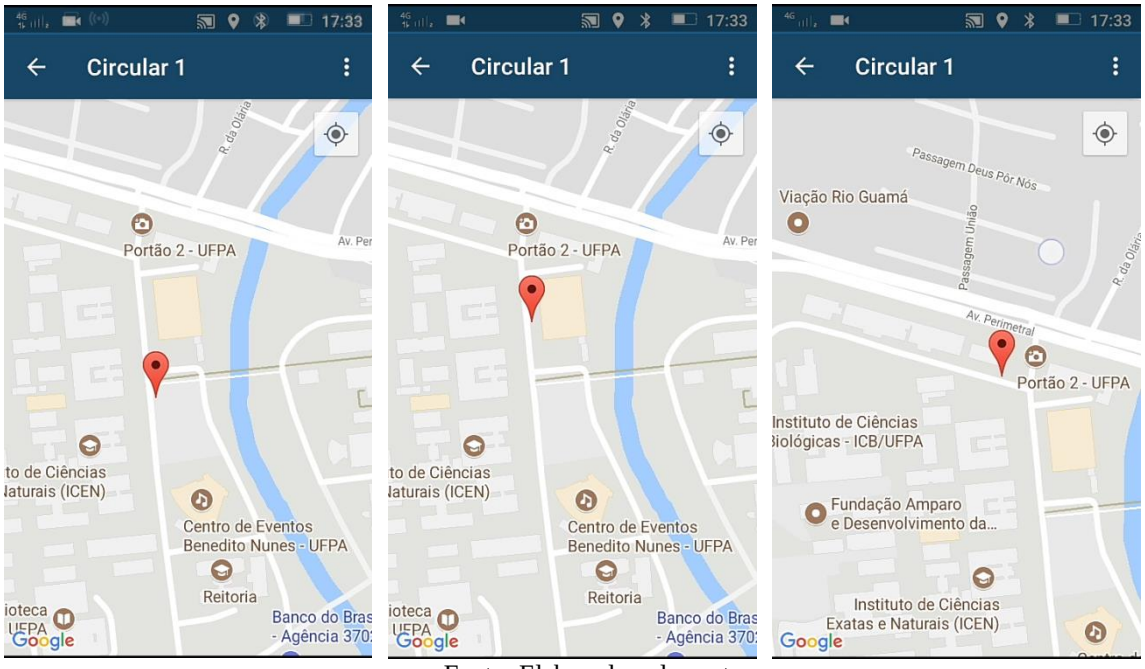
Figura 34 – Especificação dos módulos utilizados em cada *smartphone*



Fonte: Elaborado pelos autores

As imagens da Figura 35 mostram o monitoramento do "Circular 1" no módulo Usuário do aplicativo.

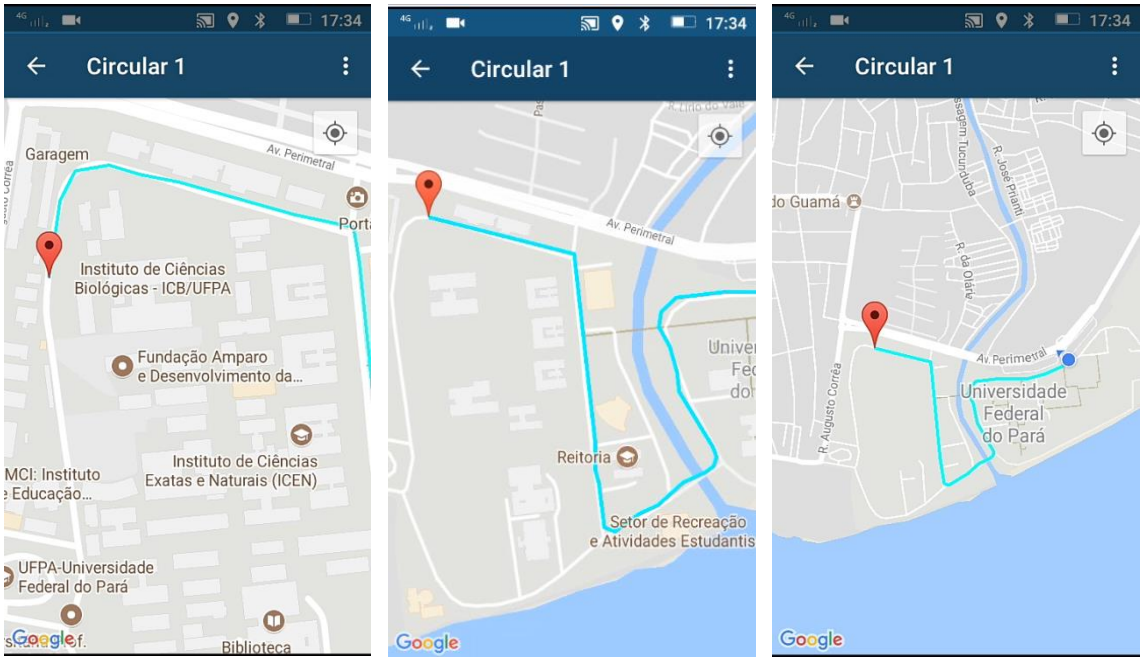
Figura 35 - Acompanhamento do dispositivo rastreado em tempo real.



Fonte: Elaborado pelos autores

As próximas imagens mostram o histórico de movimentação do mesmo.

Figura 36 - Acompanhamento do dispositivo rastreado com exibição da rota percorrida.



Fonte: Elaborado pelos autores

Além do teste de campo, foi realizado o monitoramento do funcionamento do aplicativo no próprio Android Studio, através do Android Monitor, o qual permite visualizar cada passo da execução do aplicativo através de *logs*. Sendo assim, foi possível analisar o tempo de resposta do Web Service para as principais requisições do aplicativo. A Figura 37 mostra o monitoramento de uma solicitação em um dos dispositivos utilizados.

Figura 37 – Monitoramento do Aplicativo no Android Studio

Fonte: Elaborado pelos autores

5.3. Resultados Obtidos

De modo geral, o aplicativo funcionou de maneira satisfatória, executando suas funcionalidades da forma correta nos diferentes dispositivos e versões do sistema operacional

Android que foram utilizadas, demonstrando, desse modo, a portabilidade como um ponto forte. Outro aspecto importante foi o comportamento da interface gráfica, que se ajustou de acordo com o tamanho da tela de cada dispositivo.

Quanto ao desempenho, as solicitações HTTP, foram processadas com rapidez por parte do *Web Service,* tendo uma média de tempo de resposta aceitáveis nas duas conexões de rede utilizadas, conforme mostra a Tabela 4. Em apenas um dos dispositivos observou-se certa lentidão no processamento das respostas obtidas do servidor, por exemplo, no carregamento de telas para exibição de informações ao usuário. Porém**,** esse problema está relacionado ao baixo recurso de hardware do dispositivo em questão.

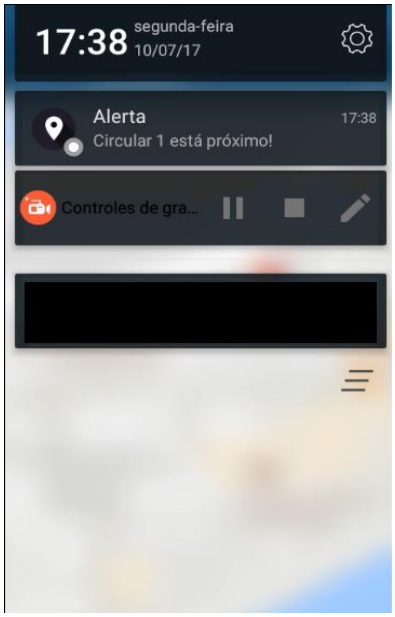
Tabela 4 – Média de tempo de resposta do *Web Service*

Tarefa	**Quantidades de Solicitações**	**Média de Tempo de Reposta em Segundos**	
		3G	**4G**
Login	10	1,037	0,81
Obter Posição	10	0,864	0,177
Enviar Posição	10	1,54	0,20
Obter Lista de Objetos	10	0,95	0,28

Fonte: Elaborado pelos autores

O módulo Rastreamento obteve a localização dos objetos com a precisão padrão do GPS que é de até 5 metros, conforme observado nas imagens anteriores do mapa retiradas do módulo Usuário. A funcionalidade de alerta de aproximação, realizou sua tarefa de maneira correta notificando o usuário da aproximação do ônibus através da vibração do dispositivo e alarme sonoro, emitidos junto com a mensagem na barra de status do mesmo. Tal funcionalidade mostrou-se muito útil, pois pode evitar que o usuário perca o Ônibus caso o mesmo se distraia. Conforme mostra a Figura 38.

Figura 38 - Notificação de aproximação



Fonte: Elaborado pelos autores

A interface foi projetada para disponibilizar as funcionalidades de maneira intuitiva, visando a facilidade de utilização por parte do usuário, porém não foram aplicados testes de usabilidade.

Dessa forma, os resultados obtidos demonstram que a aplicação atingiu de forma positiva os principais objetivos para os quais ela foi desenvolvida.

6 CONCLUSÃO

Torna-se cada vez maior a necessidade de ferramentas que mostrem em tempo real a localização de objetos ou pessoas para uma infinidade de objetivos. Neste trabalho foi desenvolvido um aplicativo que poderá ser usado para monitoramento via GPS de qualquer *smartphone*, assim como a visualização em tempo real de qualquer outro devidamente cadastrado e vinculado ao usuário.

Inicialmente, para alcançar tal objetivo, foi realizado um estudo acerca das tecnologias disponíveis que atendessem aos requisitos da aplicação, optando-se pelas gratuitas. Além disso, foram utilizadas técnicas da área de Engenharia de Software para a especificação das funcionalidades e modelagem da aplicação. Na etapa de implementação, foram utilizados os recursos da Plataforma Android, bem como tecnologias auxiliares (REST, JSON, Google Maps), a fim de adequar o código-fonte a padrões reconhecidos. Por fim, através de testes, foi possível analisar o funcionamento da aplicação levando em consideração aspectos como desempenho e portabilidade.

O desenvolvimento do aplicativo teve várias dificuldades devido aos diferentes tipos de tecnologias empregadas, como o controle do GPS, desenvolvimento do aplicativo Android, implementação do *Web Service* e comunicação entre estes diferentes sistemas.

O aplicativo se mostrou bem eficiente, já que apresentou desempenho satisfatório em *smartphones* com versões diferentes do sistema operacional Android. Apesar do tempo de resposta ser proporcional tanto ao *hardware* do *smartphone*, quanto à qualidade do sinal de GPS, os testes mostraram bom desempenho nas condições comuns da região metropolitana (intensidade do sinal e disponibilidade de rede móvel).

A utilização do aplicativo no Circular da UFPA foi satisfatória, mantendo uma cobertura de sinal constante e com baixa interferência. A previsão de ganho de tempo para os usuários é significativa, pois estes passariam menos tempo na parada esperando o Circular, já que com a utilização do aplicativo poderiam verificar o melhor momento para se deslocar até a parada.

A versatilidade deste aplicativo é ampla, podendo ele ser usado para o monitoramento de um celular em caso de furto, para rastreio de pessoas quando a segurança da mesma é um fator crítico, rastreio de veículos ou encomendas, dentre muitas outras aplicações.

O aplicativo conta com uma codificação extensível, facilitando o futuro acréscimo de funcionalidades que venham incrementá-lo ou contribuir para a criação de aplicativos mais específicos.

6.1. Trabalhos Futuros

Nesta seção é listada algumas propostas a serem inseridas na aplicação, visando a melhoria da mesma:

- ✓ Realizar testes de usabilidade para verificar a facilidade de utilização da aplicação por parte do usuário.
- ✓ Dar ao usuário a opção de escolher o período que este deseja visualizar o histórico de posições do objeto rastreado.
- ✓ Dar ao usuário a opção de definir qual a distância que este deseja ser alertado sobre a proximidade do objeto rastreado.
- ✓ Criação de classes para identificar dispositivos que estão acoplados a objetos semelhantes, podendo verificar qual destes seria o mais útil para o usuário.
- ✓ Utilização de equipamentos como *Raspberry* para aumentar as possibilidades de utilização do aplicativo.

REFERÊNCIAS

ALMEIDA, M. A. F. R. de . **LTE: Evolução das Redes Móveis**, 2013. Disponível em: <http://www.teleco.com.br/tutoriais/tutorialintlte/pagina_2.asp> Acesso em: 10 de Dezembro de 2016.

ANDROID DEVELOPES, 2017. Disponível em:<https://developer.android.com/guide/platform/index.html?hl=pt-br#art>Acesso em: 03 de Janeiro de 2017.

EGGEA, R. F. **Aplicação Android utilizando Sistema de Localização Geográfica para determinação de pontos turísticos na cidade de Curitiba**. Curitiba, 2013. Monografia (Especialização em Tecnologia Java e Desenvolvimento para dispositivos móveis). Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná.

FERREIRA, A. A. J.; GONÇALVES, W.O. **Rastreamento Veicular com auxilio de dispositivos móveis**. Brasília, 2014. Monografia (Bacharel em Engenharia Eletrônica). Universidade de Brasília.

FERREIRA, T. A.; MOREIRA, R.C.; MOZAQUATRO, P.M. Um estudo sobre computação sensível ao contexto baseado em geolocalização. **In: SEMINÁRIO INTERINSTITUCIONAL DE ENSINO, PESQUISA E EXTENSÃO**. 16., 2011, Cruz alta. Resumo. Cruz Alta: Unicruz, 2011. Disponível em: < http://www.unicruz.edu.br/16_seminario/artigos/agrarias/UM%20ESTUDO%20SOBRE%20COMPUTA%C3%87%C3%83O%20SENS%C3%8DVEL%20AO%20CONTEXTO%20BASEADO%20EM%20GEOLOCALIZA%C3%87%C3%83O.pdf>. Acesso em: 18 out. 2016.

FURTADO, A. B.; JUNIOR, J. V. C. **Prática de Análise de Projeto de Sistemas**. Belém: AB Furtado, 2010.

LÁRIOS, A. **Estudo e Construção de Cenários para a Telefonia Móvel Celular no contexto Brasileiro**. Porto Alegre, 2003. Dissertação (Mestrado em Administração). Programa de Pós-Graduação em Administração, Universidade Federal do Rio Grande do Sul.

LECHETA, R. **Google Android : Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. 2. Ed. São Paulo: Novatec, 2010.

LIMA, J. C. R. **Web Services (SOAP X REST)**. São Paulo, 2012. Monografia (Tecnólogo em Processamento de Dados). Faculdade de Tecnologia de São Paulo, FATEC-SP apud FIELDING, R. Architectural Styles and the Design of Network-based Software Architectures. Tese (Doutorado) — UNIVERSITY OF CALIFORNIA, IRVINE, 2000.

MARQUES, A. P. **Proposta de um Programa de Gestão da Qualidade para uma Empresa Genérica de Posicionamentos com GPS**. São Carlos, SP, 2006. Tese (Doutorado em Engenharia de Transportes) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos.

MONTEIRO, J. B. **Google Android: Crie aplicações para celulares e *tablets***. São Paulo: Casa do Código, 2012.

MOBILTRACKER. Disponível em:<https://www.mobiltracker.com.br> Acesso em: Dezembro, 2016.
MySQL. **Manual de Referência**. Disponível em:<https://dev.mysql.com/doc/refman/5.6/>. Acesso em: Dezembro, 2016.

PAULA, A. S. de; BRESSAN, M. A; ABE, T. **Segurança em Redes Móveis**. Curitiba, 2013. Dissertação (Pós Graduação em Redes e Segurança de Sistemas). Pontifícia Universidade Católica do Paraná.

RODRIGUES, L. C. R. **Arquitetura REST**. Juiz de Fora - MG, 2009. Monografia (Bacharel em Ciência da Computação). Departamento de Ciência da Computação, Universidade Federal de Juiz de Fora.

SAUDATE, A. **REST: Construa API's inteligentes de maneira simples**. São Paulo: Casa do Código, 2014.

SILVA, C. M. de P. e. **Utilização do Sistema de Posicionamento Global para Monitoramento de Transporte Fretado**. Campinas, SP, 2006. Dissertação (Mestrado em Engenharia Civil) – Faculdade de Engenharia Civil, Arquitetura e Urbanismo – Universidade Estadual de Campinas, UNICAMP.

SILVEIRA, I. F. **Java, das torradeiras a internet**, 2003. Disponível em:<http://www.infowester.com/lingjava.php> Acesso em: 14 de Dezembro de 2016.

SOUSA, C. R. M. **Interferidores GPS: Análise do Sistema e de Potenciais Fontes de Interferência**. Rio de Janeiro, 2005. Dissertação (Mestrado em Engenharia Elétrica). Instituto Militar de Engenharia.

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. 3. Ed. São Paulo. Pearson Prentice Hall, 2010.

KUROSE, J.F; ROSS K.W. **Redes de Computadores e a Internet: Uma abordagem top-down**. 5. Ed. São Paulo. Pearson Education, 2010.

KIELTYKA, V. SIRASS - **Sistema de Rastreamento de Smartphone**. Curitiba, 2013. Monografia (Especialização em Tecnologia Java). Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná.

W3C, 2004. Disponível em:<https://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/> Acesso em 30 de Novembro de 2016.

Smartphone OS Market Share, 2016Q. Disponível em: <http://www.idc.com/prodserv/smartphone-os-market-share.jsp> Acesso em: 24 de outubro de 2016.

UNIVERSIDADE FEDERAL DO PARÁ. **UFPA tem novos ônibus circulares para atender comunidade universitária**. 2010. Disponível em: <https://www.portal.ufpa.br/imprensa/noticia.php?cod=5011>. Acesso em: 14 de Setembro de 2016.

UNIVERSIDADE FEDERAL DO PARÁ. **Histórico e estrutura**. 2015. Disponível em: <https://www.portal.ufpa.br/includes/pagina.php?cod=historico-e-estrutura>. Acesso em: 14 set. 2016.

UOL Noticias, 2017. Disponível em: <https://tecnologia.uol.com.br/noticias/redacao/2017/10/08/ja-era-hora-celulares-terao-gps-com-precisao-de-ate-30-cm-ja-em-2018.htm. Acesso em: 09 de Outubro de 2017.