INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - IFSUL, *CAMPUS* PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

HUANTER BATISTA

GEOLOCALIZAÇÃO EM ANDROID

PASSO FUNDO
2015

HUANTER BATISTA

GEOLOCALIZAÇÃO EM ANDROID

Projeto de pesquisa submetido como requisito parcial para a aprovação na disciplina de PC II do Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, *Campus* Passo Fundo.

JOSUÉ TOEBE

PASSO FUNDO

2015

HUANTER BATISTA

GEOLOCALIZAÇÃO EM ANDROID

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Josué Toebe

_____

Rafael Marisco Bertei

_____

Alexandre Tagliari Lazzaretti

_____

Coordenação do Curso

PASSO FUNDO, 2015

RESUMO

Este projeto tem como objetivo o estudo e construção de uma aplicação destinada a plataforma Android que trabalhe com dados de localização (coordenadas geográficas). Este aplicativo deverá salvar localizações, trabalhando com API de geolocalização do Android e permitir que os mesmos sejam renderizados em um mapa, utilizando a API do Google Maps, para a visualização do local obtido. No escopo deste projeto será apresentado conceitos sobre a tecnologias usadas para a produção do aplicativo, APIs que trabalham com a localização e com o GoogleMaps (serviço de mapas). Também será analisado o aplicativo produzido com as tecnologias mencionadas no projeto, mostrando cada parte da aplicação.

Palavras-chave: Latitude, Longitude, Coordenadas, Localização, Mapas.

ABSTRACT

This project has objective study and build an application for Android platform, working with data location (geographic coordinates). This application should save locations, working with location API for Android and allow them be rendered on a map, using the Google Maps API, to visualize the location obtained. In scope this project will be presented concepts about technologies used for production application, APIs that work with the location and with the GoogleMaps (map service). Also will be analyzed the application produced with the technologies mentioned in the project, showing each part of the application.

Keywords: Latitude, Longitude, API, Coordinates, Location, Maps.

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS

API: *Application Programming Interface* (Interface de Programação de Aplicativos).
IDE: *Integrated Development Environment* (Ambiente Integrado para Desenvolvimento).
SQL: *Structured Query Language* (Linguagem de Consulta Estruturada).
MAC: Media Access Control (Controle de Acesso de Mídia).
RFID: Radio-Frequency Identification (Identificação por Radiofrequência).

SUMÁRIO

1. INTRODUÇÃO

Com o avanço tecnológico, recursos computacionais estão cada vez mais portáteis, o que antigamente só conseguíamos fazer mediante um computador, sendo ele *desktop* ou *notebook*, hoje conseguimos realizar na palma de nossas mãos com os *smartphones*.

Como pesquisas do IDC Brasil (*International Data Corporation*) preveem, o mercado de telefonia móvel cresce constantemente em nível mundial, e a tendência é que continue crescendo. Consequentemente, o mercado para programadores de dispositivos móveis também expande, oferecendo um vasto caminho para quem se especializar nesta área.

Para profissionais que viajam frequentemente, ou até mesmo para uma viajem de férias, bons restaurantes, pousadas, entre outros, são locais que marcam, fazendo com que, ou o turista ou o profissional, ao voltar para o mesmo local, procure novamente esses pontos. Como, muitas vezes, estes viajam para muitos lugares, é difícil recordar todos os pontos que chamam a atenção. O objetivo deste projeto é desenvolver um aplicativo baseado na geolocalização que irá auxiliar as pessoas nestes casos, salvando os locais favoritos de uma determinada região, e, quando o usuário retornar à cidade, será possível ver os locais que ele armazenou e a localização de cada um.

No escopo deste projeto são abordados conceitos sobre as tecnologias que foram utilizadas para a produção da aplicação descrita acima; conceitos sobre a plataforma Android, sistema operacional em que o aplicativo executará, e suas versões. Também será visto conceitos sobre Java, orientação a objetos, SQLite, geolocalização, APIs de desenvolvimento, serviços Google, e a apresentação do projeto (aplicativo), com diagramas de classe, descrição de códigos (o que eles fazem e como trabalham) e suas funcionalidades.

A principal motivação para o estudo desta tecnologia é a ausência de aplicativos que auxiliem viajantes ou turistas que gostam de registar locais de apreciação para que quando volte-se ao local, possa se saber os melhores pontos turísticos do mesmo. Outras motivações para este estudo estão relacionadas ao crescimento da área de smartphones e a familiares que exercem a função de viajante onde o aplicativo irá auxiliar em suas rotinas.

2. REFERENCIAL TEÓRICO

Neste tópico são expostos os conceitos sobre o sistema operacional Android, sobre seu funcionamento e como ele vem evoluindo ao longo dos anos. Também são abordados conceitos sobre a linguagem Java, linguagem utilizada na programação de aplicativos para Android, sobre a geolocalização e sobre a utilização e o desenvolvimento de aplicações que utilizam o recurso de localização, obtendo as coordenadas através do GPS ou redes de conexão externa (internet), e como integrar esses dados (latitude e longitude, por exemplo), obtidos através da geolocalização, com o Google Maps, assim renderizando-os em um mapa.

Por último, será apresentado a aplicação produzida com estas tecnologias, códigos, layouts, junto ao seu diagrama de classes e seus casos de uso, desenvolvidos para produzir o aplicativo.

2.1 A PLATAFORMA ANDROID

Android é um dos grandes sistemas operacionais para dispositivos móveis disponíveis em mercado hoje. Pertencente a Google, é o sistema operacional mais utilizado nos aparelhos e o que mais contém desenvolvedores na área.

Segundo o IDGNOW (2014), o sistema operacional Android predomina com 85% de participação no mercado mundial de telefonia móvel.

Grandes empresas que dominam o setor de telefonia móvel, como LG, Samsung, Sony, Motorola entre outras, optam por rodar o sistema Android em seus aparelhos; as únicas que não optam pelo sistema Android são os aparelhos da Nokia, agora comprada ela Microsoft, que rodam atualmente Windows 8.1 Mobile como sistema operante, e a Apple, que utiliza o sistema operacional IOS nos seus aparelhos, atualmente com o sistema IOS 8.1.

2.1.1 Execução de aplicações

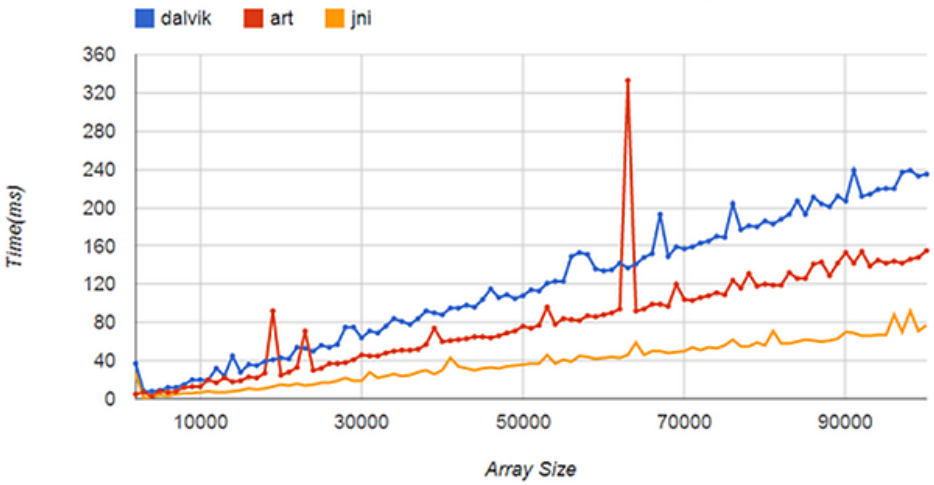
Toda e qualquer aplicação destinada à Android executa sobre uma máquina virtual denominada Dalvik. Essa máquina virtual tem semelhança com a JVM (Java Virtual Machine), máquina virtual do Java, na qual o programador produz seus códigos Java e, logo após, são traduzidos para uma linguagem binária que a máquina virtual seja capaz de entender. As aplicações feitas para Android também são produzidas

utilizando a linguagem Java, uma escolha feita pela facilidade que o Java tem com a orientação a objetos e ao controle de memória.

Segundo Souza (2014), Dalvik é uma máquina virtual, na qual o compilador traduz os códigos produzidos pelo programador em bytecodes, que são enviados para a máquina virtual Dalvik, para que a ela possa traduzir os bytecodes em um código de máquina legível para o hardware, que entenderá e executará o aplicativo. Essa tradução é feita em tempo de execução do aplicativo, uma técnica chamada *Just in Time* (JIT), o que faz o processador trabalhar constantemente. Uma das consequências dessa técnica é que, para aplicativos mais pesados, ocorre perda de desempenho, fazendo com que a tradução dos mesmos fique demorada, em comparação se o Android executasse suas aplicações de maneira nativa, e em muitos casos causando travamentos do aplicativo.

O autor fala também sobre a nova máquina virtual, que está presente na versão KitKat 4.4.4 do sistema, em fase de testes, e será implantada a versão definitiva nas versões Lollipop 5.0 e demais versões do Android, o ART, que segundo ele, utiliza o conceito de *Ahead Of Time* (AOT), que se diferencia pelo fato de executar essa tradução antes da execução do aplicativo, e não durante a execução, como a Dalvik trabalha. As vantagens de termos uma execução assim é que além dos aplicativos ficarem mais rápidos, já que o programa está todo carregado na memória, teremos um consumo de bateria reduzido, já que o processador não precisará trabalhar constantemente. Umas das desvantagens dessa técnica, é que, por economizar processamento, o aplicativo tem que ser todo carregado na memória, logo o consumo de memória aumenta, uma consequência dessa técnica de execução. Na figura 1, temos um gráfico comparando um benchmark de desempenho entre as máquinas virtuais, sendo comparado a Dalvik (azul), ART (vermelho), JNI, uma técnica que permite que um programa Java acesse aplicações e bibliotecas escritas em linguagem de menor nível, como C, C++ e Assembly (Laranja).

Figura 1: Gráfico de desempenho (benchmark) entre máquinas virtuais (Dalvik, ART e JNI)



Fonte: SOUZA, 2014

2.1.2 Versões

De acordo com o site da Android (2014), o sistema atualmente conta com 5 versões e suas subversões, sendo, no site, especificado somente da versão 1.6 Donut até as demais, que se destacam pelas características descritas abaixo:

a) **Android 1.6 *Donut*:** A característica principal desta versão foi a busca do Google e a visualização de vídeos;

b) **Android 2.0 *Éclair*:** Esta versão se destaca pela organização de telas, pastas e arquivos;

c) **Android 2.2 *FroYo*:** Nesta versão foram implantados comandos de voz, como Voice Typing, permitindo que o usuário possa fazer entradas de texto somente com a fala;

d) **Android 2.3 *Gingerbread*:** Esta versão tem como característica principal os sensores, que permitem identificar movimentos com o aparelho, uma ferramenta de grande utilidade para desenvolvedores de jogos;

e) **Android 3.0 *Honeycomb*:** Primeira versão do Android optimizada para design de tablets, sendo a característica dessa versão;

f) **Android 4.0 *Ice Cream Sandwich*:** Melhorias no layout, tornando-o mais simples, rápido e inteligente, destacaram esta versão;

g) **Android 4.1/4.2/4.3 *Jeally Bean*:** A versão Jeally Bean trouxe aos usuários o *Google Now*, que é um serviço do *Google* de informações, houve também

melhorias no layout de tablets e no serviço de localização automático através de rede *WI-FI*;

h) Android 4.4 *KitKat*: Com o lançamento da versão KitKat, o sistema de inteligência foi melhorado, principalmente em chamadas e mensagens, também foi implantado o comando de voz "*OK Google*" que permite pesquisar no buscador sem a necessidade de digitar.

i) Android 5.0 *Lollipop*: Versão atual do sistema Android, que traz mudanças em praticamente todo o layout, movimentação suave e dinâmica. Também houve melhorias em consumo de bateria, segurança, aproveitando mais do hardware do dispositivo com menos consumo de energia e maior interatividade e controle do smartphone por parte do usuário.

2.2 JAVA

A linguagem Java foi feita para ser uma linguagem pequena, portátil, robusta e orientada a objetos. Por tais características foi a "linguagem escolhida" para a World Wide Web (WWW), que pode ser acessada de diversas plataformas e sistemas operacionais (DOHERTY MANNING, 1998, p. 10).

> O Java foi projetado para ser simples, orientado a objetos, distribuído, seguro, portável, interpretado, de arquitetura neutra, robusto, de alto desempenho, multithread e dinâmico (SUN, 2014).

Java se caracteriza por ser fortemente orientado a objetos. Conforme Doherty e Manning (1998), Java é a verdadeira linguagem orientada a objeto que permite você modular programas de forma flexível. Os autores ainda comentam sobre suas funcionalidades, compostas de um conjunto de bibliotecas e classes. A linguagem também oferece protocolo de internet, manipulação de imagens, tal função vista no jogo Minecraft, que manipula seus personagens através de imagens PNG, além de oferecer toolkits e a possibilidade de ampliar essas funções conforme a necessidade do programador.

2.2.1 Uma breve história sobre o Java

Java foi idealizado na década de 90, pela antiga Sun Microsystems, sendo a Sun Microsystems, em 2010, comprada pela Oracle, sendo hoje a empresa que mantém a linguagem.

Java primeiramente não teve a ideia inicial de ser uma linguagem de programação, e sim uma interação entre aparelhos eletrônicos. Segundo Doherty e Manning (1998), o Java inicialmente foi produzido para resolver problemas pessoais, relacionados a equipamentos, como micro-ondas, fornos, torradeiras, entre outros. Nenhuma das linguagens disponíveis na época eram robustas, pequenas e portáteis, então a ideia do Java inicialmente era atender esses requisitos para que os equipamentos tivessem uma forma de comunicação entre si.

A ideia da linguagem ser uma assistente digital falhou, e então a empresa Sun Microsystems, dona da linguagem na época, viu que o Java seria ideal para outra finalidade, a programação Web.

A linguagem ficou pertencente a Sun até 2010, quando a empresa foi comprada pela Oracle, que hoje mantém a linguagem. “Oracle adquiriu a Sun Microsystems em 2010, e desde então os engenheiros de hardware e software da Oracle têm trabalhado lado a lado para construir sistemas totalmente integrados e soluções otimizadas, destinadas a alcançar os níveis de desempenho que são incomparáveis na indústria” (ORACLE, 2014).

2.3 GEOLOCALIZAÇÃO

Segundo Karasinski (2010), quando estamos com algum equipamento que contenha recursos computacionais e este esteja conectado com a internet, é possível obter as coordenadas geográficas através da identificação do IP (endereço que a máquina recebe ao estar conectada na internet), assim obtendo o país, estado, cidade, local e hora independentemente de onde estivermos.

O autor também fala sobre a geolocalização, sem estar conectado à internet, que pode ser utilizada com dados a partir de um endereço MAC, RFID, conexão sem-fio e coordenadas de um GPS. Vários smartphones utilizam o GPS integrado para enviar as informações de localização. Alguns, como o iPhone, pedem a sua permissão antes (KARASINSKI, 2010).

2.3.1 Geolocalização no Android

Após entender um pouco sobre o que é a geolocalização, será analisado como a geolocalização funciona na plataforma Android. Segundo Pereira (2014), a localização no Android é feita de 3 formas:

a) **Redes móveis:** Android suporta sensores que determinam a localização via triangulação com redes móveis;
b) **WI-FI:** Android suporta provedores de localização, que captam as coordenadas via rede WI-FI;
c) **Geo-Positioning-System (GPS):** Android suporta sensores que captam sinais de satélite, obtendo as coordenadas.

2.3.1.1 Android Location

Android Location é uma API disponibilizada pela Google para desenvolvimento de aplicações que utilizem o recuso de localização. Esta API conta com vários recursos, como interfaces, classes e métodos para auxiliar o programador a desenvolver seu aplicativo.

2.3.1.2 Location, LocationListener e LocationManager

Segundo Pereira, em se tratando de programação para a API Android Location, implementa-se uma interface chamada LocationListener e instancia-se um objeto do tipo LocationManager, assim seguindo boas práticas de programação.

Conforme o site Android Developers (2014), quando é implementada a interface, ela obriga a implementação de 4 métodos, provenientes da interface, os quais são citados e descritos abaixo:

a) ***onLocationChanged***: este método é chamado quando o local for alterado;
b) ***onProviderDisabled***: este método é chamado quando o provedor é desativado pelo usuário;
c) ***onProviderEnabled***: este método é chamado quando o provedor é ativado pelo usuário;
d) ***onStatusChanged***: este método é chamado quando altera-se o estado do provedor.

Após esta programação feita, a próxima etapa é instanciar o LocationManager, que vai trabalhar diretamente com a localização, fazendo requisições, entre outras funções.

Este objeto conta com inúmeros métodos públicos, métodos que trabalham com requisições, status do GPS, entre outras funcionalidades. Abaixo destaca-se alguns dos principais métodos:

a) ***getAllProviders***: retorna uma lista de todos os provedores conhecidos;
b) ***getGpsStatus***: recupera informações sobre o estado atual do GPS;
c) ***removeUpdates***: remove todas as atualizações de localização para o LocationListener especificado;
d) ***getProviders***: retorna uma lista com os nomes dos fornecedores de localização.
e) ***requestLocationUpdates:*** Faz as requisições de localização a partir da interface LocationListener.

Ainda conta-se com constantes, que, conforme Android Developers (2014), são variáveis que auxiliam na busca de informações, em que pode-se especificar qual a tecnologia a ser usada, para obter os dados de localização. Destacam-se algumas:

a) ***GPS_PROVIDER***: provedor que determina a localização por meio de satélites;
b) ***NETWORK_PROVIDER***: provedor que determina a localização por meio de redes, disponível para redes móveis (2G, 3G [...]) e redes WI-FI.

Ainda nesta parte, são oferecidos recursos pelo objeto Location, trabalhando em conjunto com o LocationManager, que é o objeto que trabalha com os dados das coordenadas. Ele contém alguns métodos importantes para o desenvolvimento de aplicações, descritos abaixo:

a) ***getLatitude***: obtém os dados da latitude em que se encontra o aparelho;
b) ***getLongitude***: obtém os dados da longitude em que se encontra o aparelho;
c) ***getTime***: obtém os dados de hora, dia, mês e ano deste 01 de janeiro de 1970;
d) ***getAltitude***: obtém os dados da altitude em que se encontra o aparelho.

O site ainda resume a função que cada um tem e como eles trabalham. Conforme o site, o LocationListener controla quando o local for alterado, guardando as informações dos locais recebidos, informações que ele recebe através do LocationManager, classe que controla as funções de busca de localizações e coordenadas através do objeto Location.

2.4 GOOGLE MAPS

Segundo site TECHTUDO (2014), Google Maps é o mapa online do Google disponível para Android e para iOS, uma ferramenta excelente para encontrar qualquer lugar no mundo, obter instruções de como ir de um lugar para outro e caminhar ao redor das cidades.

2.4.1 Integrando a geolocalização com o Google Maps

Para integrar os dados obtidos com a geolocalização, ao Google Maps, primeiramente necessitamos de uma chave de acesso, para que se possa utilizar o serviço de mapa, tal chave é obtida com a própria Google. Segundo a empresa, essa chave serve para verificar a participação no desenvolvimento da Google, requerida cada vez que o aplicativo solicitar dados dos serviços Google.

2.4.1.1 MapView

MapView é uma classe é trabalha com o layout e visualização do mapa, manipulando ações do usuário, como ampliar o mapa, e marcação de coordenadas, como os dados que são obtidos através da geolocalização.

> A principal classe na biblioteca do Google Maps é MapView, uma subclasse de ViewGroup, na biblioteca padrão do Android. A classe MapView exibe um mapa com dados obtidos a partir do serviço do Google Maps. Quando a classe MapView está em foco, ela captura pressionamento de teclas e gestos de toque para fazer panorâmicas e ampliar o zoom do mapa automaticamente, incluindo a manipulação de solicitações de rede para blocos de mapas adicionais. Essa classe também fornece todos os elementos da interface de usuário necessários para que os usuários controlem o mapa (Google Developers, 2014).

Para obter-se o mapa é usado o método da classe pai (ViewGroup), chamado getMap, assim iniciando automaticamente os pontos e vistas do mapa.

2.4.1.2 Google Maps API v2

Google Maps API v2 é uma API destinada a renderizações de mapas, utilizando o Google Maps.

> Com a API do Google Maps para Android, você pode adicionar mapas com base em dados do Google Maps para a sua aplicação. A API trabalha automaticamente com acesso aos servidores do Google Maps, download de dados, visualização do mapa, e resposta para mapear gestos. Você também pode usar chamadas de API para adicionar marcadores, polígonos e sobreposições para um mapa básico, e para mudar a visão do usuário de uma determinada área do mapa (Google Developers, 2014).

Esta API trabalha com fragmentos, que são adicionados no layout da aplicação. Após adicionado o fragmento no layout o mesmo deve ser obtido pela classe que controla o layout, assim podemos manipular o mapa da forma que quisermos.

Nesta API temos funções, objetos e classes que trabalham para montar o mapa, das quais se destacam:

a) ***LatLng***: define os valores de latitude e longitude para adicionarmos ao mapa;
b) ***addMarker***: adiciona um ponto de marcação nas coordenadas definidas pelo LatLng;
c) ***moveCamera***: movimenta o mapa;
d) ***animateCamera***: ajusta o zoom da visualização do ponto.

2.5 SERVIÇOS GOOGLE

2.5.1 Google play

Google play ou Google play Store é a loja virtual da Google que possibilita ao usuário do sistema Android baixar suas aplicações, músicas, filmes, livros, entre outros, de forma gratuita ou paga. Antigamente existia uma loja virtual de aplicativos para Android, chamada Android Market, que foi substituída pela atual Google Play Store; usuários antigos do sistema que não fizeram a atualização ou não entraram na

loja ainda, ainda podem se deparar com o ícone do Android Market no *display* do dispositivo (TAVARES, 2012).

2.5.2 Google Play Service

Google Play Service é um serviço da Google que serve para gerenciar aplicativos instalados no dispositivo. Segundo a Google Play (2014), é um componente essencial para fazer autenticação para serviços do Google, contatos sincronizados, acesso a todas as configurações de e serviços baseados na localização com maior qualidade e menor consumo de recursos.

2.6 SQLITE

Conforme o site SQLite (2014), “SQLite é um banco de dados embutido. Diferentemente da maioria dos outros bancos de dados, o SQLite não tem um processo de servidor separado, ele lê e escreve diretamente para arquivos de disco comuns.”

Para acessar o SQLite e seus dados na plataforma Android, primeiramente deve-se ter um shell remoto, para que se possa acessar o arquivo do banco e a partir deste ponto iniciar os comandos do SQLite.

“SQLite é um banco de dados SQL completo com várias tabelas, índices, gatilhos e pontos de vista, está contido em um único arquivo em disco.” (SQLITE, 2014).

3. PROJETO DO APLICATIVO

O aplicativo foi desenvolvido utilizando a IDE do Android para desenvolvimento (Android Studio) e, para testes, foi utilizado o emulador SDK e um aparelho real.

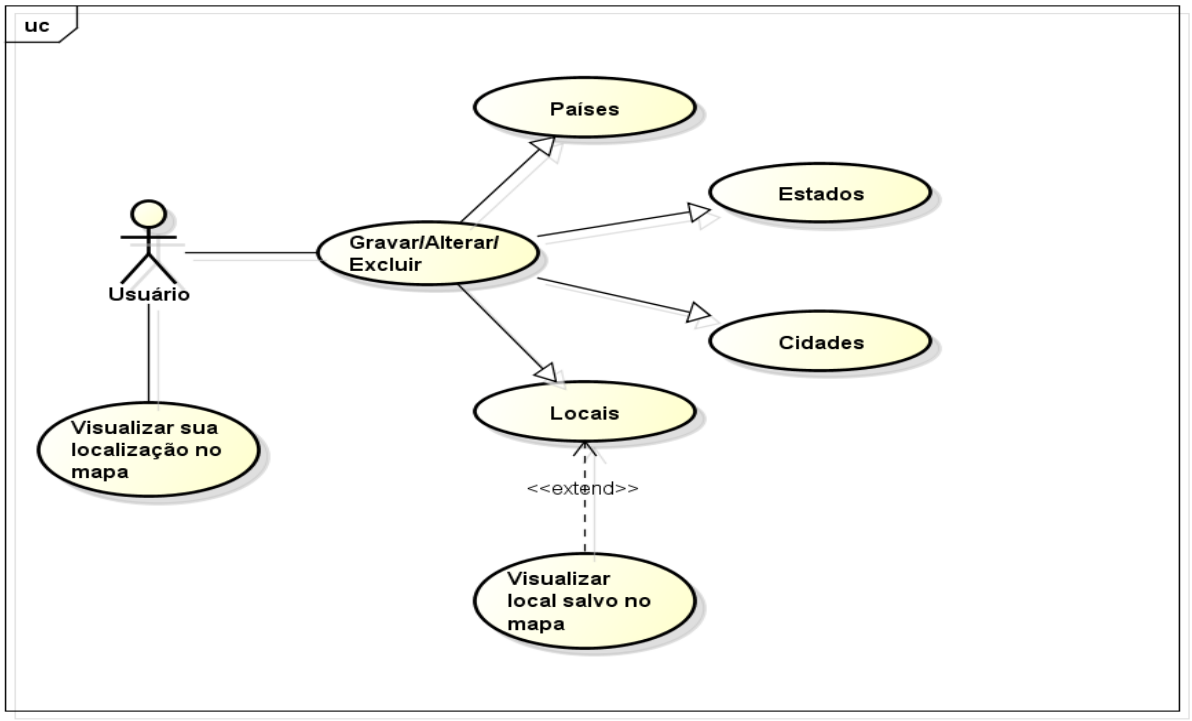
Dando continuidade ao projeto, foi analisado como a plataforma trabalha, questões de chaves de acesso, layout, permissões, como funciona a geolocalização na plataforma Android, obtendo coordenadas geográficas e como integrar esses dados ao Google Maps.

Por último, foi analisado o funcionamento da aplicação, se os dados obtidos de localização estavam corretos e se os mesmos interagiam com o Google Maps, assim sendo mostrado o mapa com as coordenadas no display do dispositivo.

3.1 DIAGRAMA DE CASOS DE USO

Na figura 2, é apresentado o diagrama de casos de uso. Neste diagrama estão apresentadas as possíveis ações que o usuário pode fazer executando a aplicação com o menu de opções, que faz o manuseio da aplicação, disponível ao mesmo.

Figura 2: Diagrama de casos de uso da aplicação



Fonte: Autor.

Após executar o aplicativo, o usuário poderá ver a sua localização atual renderizada em um mapa, e também terá acesso a um menu lateral, onde ele poderá acessar os dados salvos no aplicativo (países, cidades, estados e locais). A partir da tela de listagem, o usuário poderá fazer ações de cadastro dos registros (excluir, incluir alterar) e, caso a listagem seja das localizações, ele poderá visualizar o registro salvo em um mapa.

O caso de uso gravar/alterar/excluir permite ao usuário, manusear a aplicação conforme a necessidade, salvando o país que desejar, um estado referente a um país, uma cidade referente a um estado e um local referente a uma cidade.

O caso de uso visualizar sua localização no mapa, permite ao usuário a visualização de sua localização atual no mapa.

O caso de uso visualizar local salvo no mapa, permite ao usuário a visualização de seu ponto salvo, e clicando em cima do marcador exibido no mapa ele poderá navegar de onde ele se encontra, até o ponto salvo, usando a navegação do Google Maps (Versão beta).

3.1.1 Documentação de casos de uso

Neste tópico são exemplificados os casos de uso mostrados na figura 2, o ator principal que irá interagir com esses casos de uso, quais são as condições para utilização dos casos, e quais são as opções disponibilizadas se o usuário escolher uma determinada opção.

A. Caso de Uso: Visualizar sua localização no mapa

A documentação abaixo exemplifica este caso de uso.

Tabela 1: Caso de uso: Visualizar sua localização no mapa

Nome do caso de uso	Visualizar sua localização no mapa.
Ator principal	Usuário.
Pré-condições	Estar com a aplicação iniciada.
Usuário executa a aplicação	Seus dados de localização são obtidos através do GPS ou Internet e são renderizados no mapa disponível nesta tela.
Quando o usuário não estiver com o GPS ou Internet disponível	Será mostrado uma mensagem no início da aplicação para que o usuário ative pelo menos um provedor de dados.
Quando não é possível obter as coordenadas geográficas	Será avisado ao usuário que não foi possível obter a sua localização, para que o mesmo utilize a aplicação em um ambiente mais aberto (caso o provedor seja GPS), para que os dados possam ser obtidos.

B. Caso de Uso: Gravar/Alterar/Excluir Países

A documentação abaixo exemplifica este caso de uso.

Tabela 2: Caso de uso: Gravar/Alterar/Excluir Países

Nome do caso de uso	Gravar/Alterar/Excluir Países.
Ator principal	Usuário.
Pré-condições	Estar com a aplicação iniciada e com o menu lateral disponível.
Usuário seleciona no menu lateral a opção países	Será mostrada uma listagem dos países salvos pelo usuário.
Usuário seleciona no menu a opção adicionar novo registro	Será mostrado um formulário para cadastrar um novo país.
Usuário escolhe um registro na lista e clica e segura no mesmo	Será mostrado um menu alternativo com as opções de alterar o registro ou excluir. Caso a opção seja alterar, será mostrado o mesmo formulário com os dados para edição.

C. Caso de Uso: Gravar/Alterar/Excluir Estados

A documentação abaixo exemplifica este caso de uso.

Tabela 3:Caso de uso: Gravar/Alterar/Excluir Estados

Nome do caso de uso	Gravar/Alterar/Excluir Estados.
Ator principal	Usuário.
Pré-condições	Estar com a aplicação iniciada e com o menu lateral disponível.
Usuário seleciona no menu lateral a opção estados	Será mostrada uma listagem dos estados salvos pelo usuário.
Usuário seleciona no menu a opção adicionar novo registro	Será mostrado um formulário para cadastrar um novo estado.
Usuário escolhe um registro na lista e clica e segura no mesmo	Será mostrado um menu alternativo com as opções de alterar o registro ou excluir. Caso a opção seja alterar, será mostrado

	o mesmo formulário com os dados para edição.
Caso não tenha nenhum país salvo	Será informado ao usuário que cadastre pelo menos um país.

D. Caso de Uso: Gravar/Alterar/Excluir Cidades

A documentação abaixo exemplifica este caso de uso.

Tabela 4: Caso de uso: Gravar/Alterar/Excluir Cidades

Nome do caso de uso	Gravar/Alterar/Excluir Cidades.
Ator principal	Usuário.
Pré-condições	Estar com a aplicação iniciada e com o menu lateral disponível.
Usuário seleciona no menu lateral a opção cidades	Será mostrada uma listagem das cidades salvas pelo usuário.
Usuário seleciona no menu a opção adicionar novo registro	Será mostrado um formulário para cadastrar uma nova cidade.
Usuário escolhe um registro na lista e clica e segura no mesmo	Será mostrado um menu alternativo com as opções de alterar o registro ou excluir. Caso a opção seja alterar, será mostrado o mesmo formulário com os dados para edição.
Caso não tenha nenhum estado salvo	Será informado ao usuário que cadastre pelo menos um estado.

E. Caso de Uso: Gravar/Alterar/Excluir Locais

A documentação abaixo exemplifica este caso de uso.

Tabela 5: Caso de uso: Gravar/Alterar/Excluir Locais

Nome do caso de uso	Gravar/Alterar/Excluir Locais.
Ator principal	Usuário.
Pré-condições	Estar com a aplicação iniciada e com o menu lateral disponível.

Usuário seleciona no menu lateral a opção locais	Será mostrada uma listagem dos locais salvos pelo usuário.
Usuário seleciona no menu a opção adicionar novo registro	Será mostrado um formulário para cadastrar um novo local.
Usuário escolhe um registro na lista e clica e segura no mesmo	Será mostrado um menu alternativo com as opções de alterar o registro ou excluir. Caso a opção seja alterar, será mostrado o mesmo formulário com os dados para edição.
Caso não tenha nenhuma cidade salva	Será informado ao usuário que cadastre pelo menos uma cidade.
Quando o usuário não estiver com o GPS ou Internet disponível	Será mostrado uma mensagem no início da aplicação para que o usuário ative pelo menos um provedor de dados.
Quando não é possível obter as coordenadas geográficas	Será avisado ao usuário que não foi possível obter a sua localização, para que o mesmo utilize a aplicação em um ambiente mais aberto (caso o provedor seja GPS), para que os dados possam ser obtidos.

F. Caso de Uso: Visualizar local salvo no mapa

A documentação abaixo exemplifica este caso de uso.

Tabela 6: Caso de uso: Visualizar local salvo no mapa

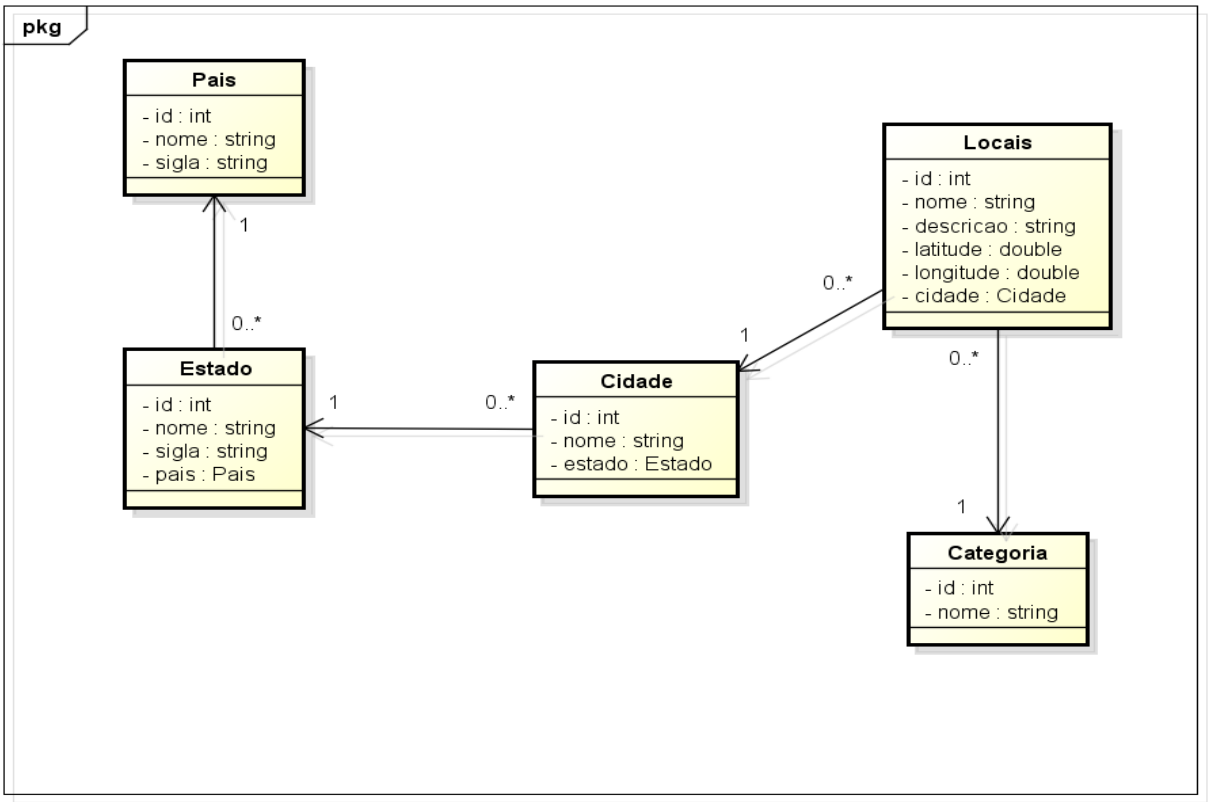
Nome do caso de uso	Visualizar local salvo no mapa
Ator principal	Usuário
Pré-condições	Ter pelo menos um local salvo com suas devidas coordenadas e selecionar a opção no menu alternativo.
Usuário seleciona no menu alternativo a opção visualizar no mapa.	**a)** Abre um mapa com a visualização do ponto salvo pelo usuário.

	b) Contém 2 opções provenientes da classe do mapa, uma para visualizar o ponto no Google Maps e outra para navegar até o ponto salvo através da navegação do Google Maps (versão beta).
Quando o usuário não estiver conectado à internet	A navegação estará indisponível e uma mensagem será mostrada ao usuário com esta informação.

3.2 DIAGRAMA DE CLASSES

Neste tópico é apresentado o diagrama de classes da aplicação, quais são as classes da aplicação, quais tabelas contém o banco de dados, quais são seus atributos, os quais, o usuário deverá preencher na hora de gravar um registro.

Figura 3: Diagrama de classes



Fonte: Autor.

Na figura 3, é exposto o diagrama de classes que contém 4 classes (país, estado, cidade e locais), e, consecutivamente 4 tabelas no banco de dados.

3.2.1 Documentação do diagrama de classes

Nesta seção é explicado as classes do diagrama, descrevendo seus atributos e métodos e como elas interagem com a aplicação.

A. Classe País

A documentação abaixo exemplifica esta classe do diagrama.

Tabela 7: Diagrama de classes: Classe País

Nome da classe	País.
Descrição	Classe responsável por dados relacionados a um país.
Atributos	**a)** ***Integer id:*** id do país. **b)** ***String nome:*** nome do país. **c)** ***String sigla:*** sigla do país.
Métodos	**a)** ***Geters e seters:*** métodos para definir e obter valores dos atributos. **b)** ***toString:*** método para exibir dados quando a classe for solicitada.

B. Classe Estado

A documentação abaixo exemplifica esta classe do diagrama.

Tabela 8: Diagrama de classes: Classe Estado

Nome da classe	Estado.
Descrição	Classe responsável por dados relacionados a um estado.
Atributos	**a)** ***Integer id:*** id do estado. **b)** ***String nome:*** nome do estado. **c)** ***String sigla:*** sigla do estado.

	d) ***Pais país:*** objeto país para este estado.
Métodos	**a)** ***Geters e seters:*** métodos para definir e obter valores dos atributos. **b)** ***toString:*** método para exibir dados quando a classe for solicitada.

C. Classe Cidade

A documentação abaixo exemplifica esta classe do diagrama.

Tabela 9: Diagrama de classes: Classe Cidade

Nome da classe	Cidade.
Descrição	Classe responsável por dados relacionados a uma cidade.
Atributos	**a)** ***Integer id:*** id da cidade. **b)** ***String nome:*** nome da cidade. **c)** ***Estado estado:*** objeto estado para esta cidade.
Métodos	**a)** ***Geters e seters:*** métodos para definir e obter valores dos atributos. **b)** ***toString:*** método para exibir dados quando a classe for solicitada.

D. Classe Locais

A documentação abaixo exemplifica esta classe do diagrama.

Tabela 10: Diagrama de classes: Classe Locais

Nome da classe	Locais.
Descrição	Classe responsável por dados relacionados a um local.
Atributos	**a)** ***Integer id:*** id do local. **b)** ***String nome:*** nome do local.

	c) ***String descrição:*** descrição do local. **d)** ***Double latitude:*** latitude do local a ser salvo. **e)** ***Double longitude:*** longitude do local a ser salvo. **f)** ***Cidade cidade:*** objeto cidade para este local
Métodos	**a)** ***Geters e seters:*** métodos para definir e obter valores dos atributos. **b)** ***toString:*** método para exibir dados quando a classe for solicitada.

E. Classe Categoria

A documentação abaixo exemplifica esta classe do diagrama.

Tabela 11: Diagrama de classes: Classe Categoria

Nome da classe	Categoria.
Descrição	Classe responsável por dados relacionados a categoria de um local.
Atributos	**a)** ***Integer id:*** id da categoria. **b)** ***String nome:*** nome da categoria.
Métodos	**c)** ***Geters e seters:*** métodos para definir e obter valores dos atributos. **d)** ***toString:*** método para exibir dados quando a classe for solicitada.

3.3 APLICATIVO GERENCIADOR DE PONTOS TURÍSTICOS

Nesta seção é exposto o aplicativo (protótipo) produzido com as tecnologias citadas acima, com layout, códigos (o que eles fazem, qual sua função), e imagens das telas para analisar como ficou a organização dos códigos e layout na tela do dispositivo.

3.3.1 Android Manifest

Antes de começar a produzir a aplicação, é necessário definir algumas permissões para o aplicativo para que o mesmo possa utilizar os serviços de GPS, classes do Google e conexão com a internet. Estas permissões devem ser especificadas no arquivo AndroidManifest, que trata desta parte. Na figura 4 temos as permissões especificadas no arquivo e quais são suas funções.

Figura 4: AndroidManifest (permissões para a aplicação utilizar recursos do smartphone)

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" /> <!-- permissão de acesso a dados de localização -->
<uses-library android:name="com.google.android.maps" /> <!-- biblioteca do google mas -->
<uses-permission
    android:name="android.permission.ACCESS_COARSE_LOCATION" /> <!-- permissão de acesso a dados de localização -->
<uses-permission android:name="android.permission.INTERNET" /> <!-- permissão de acesso à internet -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" /> <!-- permissão de acesso a rede externa -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" /> <!-- exigida pela classe do mapa -->
<uses-permission
    android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" /> <!-- permissão para buscar dados dos provedores
```

Fonte: Autor.

3.3.2 Mapa, coordenadas e GoogleMap

Quando o usuário executa a aplicação, ele pode ver a sua localização atual marcada em um mapa. Para fazer este mapa foi utilizada a API Google Maps v2, por conter mais recursos e facilidades para manuseá-la, além de conter opções de navegação do Google Maps (ANDROID DEVELOPERS, 2015).

Para utilizar essa API é preciso de uma activity (uma tela), ou seja, ter uma classe Java e um arquivo xml referente ao layout. No xml foi definido um fragmento que corresponde ao fragmento do Google Maps, e dentro do fragmento foi adicionado o nome da classe que o fragmento irá trabalhar, no caso o mapa.

Figura 5: Fragmento (XML) para montar o mapa

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginBottom="22dp"
    android:id="@+id/map"
    tools:context="ifsul.edu.br.geolocalizacaosistema.MenuLateral"
    android:name="com.google.android.gms.maps.SupportMapFragment" />
            <!-- Nome da classe referente ao mapa -->
```

Fonte: Autor.

Após esta etapa, é necessário obter este fragmento na classe que controla o xml, sendo obtido da seguinte maneira:

Figura 6: Obter o fragmento do mapa

```
mapa = ((SupportMapFragment) getSupportFragmentManager().
        findFragmentById(R.id.map)).getMap(); // mapa type = GoogleMap
```

Fonte: Autor.

Depois de obter o fragmento do mapa na classe, já pode-se manipulá-lo como desejar. Para mostrar a localização atual do usuário no mapa foi definido dados de localização, latitude e longitude, utilizando a classe LatLng e adicionando a mesma ao mapa, junto a um marker (marcador de localização), para apontar onde as coordenadas referenciam no mapa. Também podemos controlar a câmera e o zoom que será aplicado no mapa, para mostrar o ponto. A figura 7 exemplifica o código referente a esta parte, na aplicação.

Figura 7: Definindo detalhes do mapa (Marcadores, Zoom, movimentação)

```
local = new LatLng(latitude, longitude); // classe que define as coordenadas geográficas
mMap.addMarker(new MarkerOptions()// adiciona um marcador no mapa
        .position(local) // posição do marcador
        .title("Seu ponto salvo"));  // título para o marcador
mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(local, 30)); // adiciona um zoom no ponto salvo no mapa
mMap.animateCamera(CameraUpdateFactory.zoomTo(10), 2000, null); // faz a rotação do mapa no ponto salvo
```

Fonte: Autor.

Para pegar os dados das coordenadas geográficas, foi implementado, na classe que controla o mapa, a interface *LocationListener*, que trabalha com a busca dos dados de latitude e longitude, e também foi instanciado um objeto do tipo LocationManager, que irá controlar as requisições de localização, como o provedor que irá utilizar, o tempo e a distância para atualizar a localização do usuário.

Figura 8: Obtendo a localização (Serviços, requisições e parâmetros)

```
locationManager = (LocationManager)
        getSystemService(LOCATION_SERVICE); // pega o servico local de localizações
locationManager.requestLocationUpdates(
        LocationManager.GPS_PROVIDER, // provedor (GPS_PROVIDER ou NETWORK_PROVIDER)
        30000, // tempo em milissegundos para atualizar a localização
        5, // distância em metros para atualizar a localização
        this // LocationListener implementado nesta classe
); // faz requisições de localização, conforme os parêmetros
```

Fonte: Autor.

Quando é implementado a interface LocationListener ela obriga a implementação de métodos obrigatórios, um destes métodos é responsável pela busca da localização, sempre que alterar a mesma conforme os parâmetros especificados no método da figura 8, o método *onLocationChanged* que contém como parâmetro um objeto do tipo Location que faz a busca pelos dados de latitude e longitude.

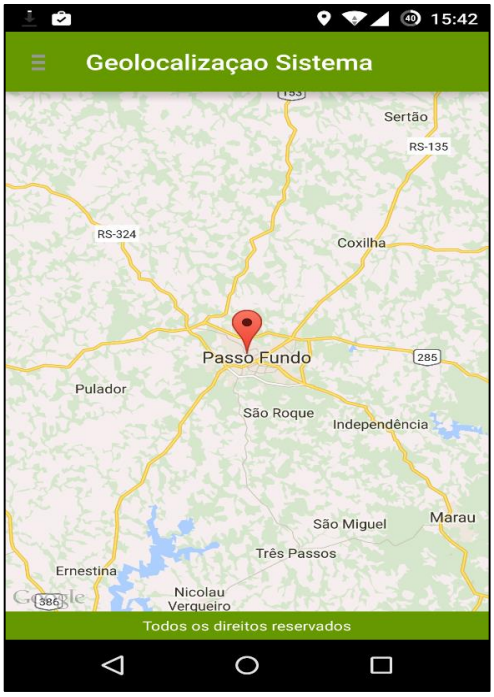
Figura 9: Obtendo coordenadas geográficas e adicionando-as ao mapa

```
@Override
public void onLocationChanged(Location location) {
    local = new LatLng(location.getLatitude(), location.getLongitude());
    if (mapa == null) {
        mapa = ((SupportMapFragment) getSupportFragmentManager().
                findFragmentById(R.id.map)).getMap();
    } else {
        mapa.clear();
    }
    markerOptions = new MarkerOptions().position(local).title("Sua Localização");
    mapa.addMarker(markerOptions);
    mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(local, 20));
    mapa.animateCamera(CameraUpdateFactory.zoomTo(10), 2000, null);
}
```

Fonte: Autor.

A figura 9 mostra a busca de dados de latitude e longitude com o objeto Location e já integrando esses dados com o mapa, sendo atualizado conforme os parâmetros do método *requestLocationUpdates.* Na figura 10 temos o layout do mapa na tela do dispositivo.

Figura 10: Imagem do mapa na tela inicial



Fonte: Autor.

3.3.3 Menu lateral

Após mostrar a localização atual renderizada em um mapa, foi produzido um menu lateral, para que o usuário possa acessar as opções de cadastro (classes do diagrama) dos registros.

Para gerar este menu, é necessário criar uma activity do tipo *Navigation Drawer,* opção disponível na IDE do Android Studio. Após criada ela gerará cinco arquivos, duas classes Java e três arquivos xml de layout, onde é montado um menu lateral simples já utilizável. Na aplicação foi alterado o layout do menu lateral e chamada do mesmo na aplicação, para ficar um layout mais elegante para o usuário. Para fazer esta mudança foi alterado o arquivo *fragment_navigation_drawer.xml,* criado automaticamente pela IDE, e adicionado um novo layout, modificando o antigo que era somente uma lista. Após esta mudança, é necessário alterar o arquivo *NavigationDrawerFragment*, também criado pela IDE, que controla o xml alterado, nesta classe alteramos o método *onCreateView*, para ele buscar nosso novo layout. As figuras 11 e 12 (XML e Java) demonstram as mudanças feitas no menu lateral.

Figura 11: Criando o Menu lateral (layout)

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/white"
    tools:context="ifsul.edu.br.geolocalizacaosistema.NavigationDrawerFragment">

    <TextView...>

    <TextView...>

    <ListView...>

</RelativeLayout>
```

Fonte: Autor.

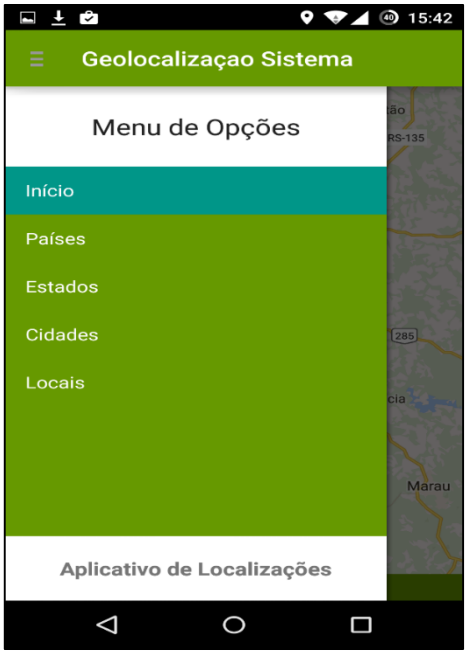
Figura 12: Criando o Menu lateral (classe)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_navigation_drawer, container, false);
    mDrawerListView = (ListView)
            view.findViewById(R.id.navigationItems); // pegando a lista do layout
    mDrawerListView.setOnItemClickListener((parent, view, position, id) -> {
            selectItem(position);
    }); // método para quando o usuário clicar em um item da lista
    mDrawerListView.setAdapter(new ArrayAdapter<String>(
            getActionBar().getThemedContext(),
            android.R.layout.simple_list_item_activated_1,
            android.R.id.text1,
            new String[]{
                    "Início",
                    "Países",
                    "Estados",
                    "Cidades",
                    "Locais"
            })); // adicionando itens a lista
    mDrawerListView.setItemChecked(
            mCurrentSelectedPosition,
            true); // verificando o item selecionado pelo usuário
    return view;
}
```

Fonte: Autor.

Após concluir esta etapa, o usuário já tem disponível o meu para uso, abaixo temos a demonstração do menu lateral.

Figura 13: Imagem do menu lateral



Fonte: Autor.

3.3.4 Listagem de dados e ContextMenu

Quando o usuário seleciona uma seção no meu lateral, ele tem acesso a uma lista que lhe mostrará os dados, conforme o solicitado. Esta listagem foi produzida com base na busca de registros no banco de dados a partir de um Cursor, onde a classe identifica o tipo de dado que o usuário deseja acessar e busca todos os registros no banco referentes a esse dado. Para que esses dados sejam exibidos ao usuário, foi criado uma lista no layout, lista semelhante as demais apresentadas no projeto. O código que faz a busca de dados está especificado na figura 14.

Figura 14: Busca de registros no banco de dados

```
public Cursor listarTodos(String ordem, String tabela){
    return getReadableDatabase().rawQuery("SELECT * FROM " + tabela + " ORDER BY " + ordem, null);
}

public Cursor listarEspecifico(String tabela, Integer id){
    return getReadableDatabase().rawQuery("SELECT * FROM " + tabela + " WHERE _id = "+id, null);
}
```

Fonte: Autor.

Conforme a necessidade do aplicativo ele escolherá se deseja listar todos os dados ou listar somente um dado específico (ex.: buscar um estado de uma determinada cidade).

Após isto, os dados obtidos foram adicionados à lista, cuja precisa de um adaptador para ser exibida e de um layout para adaptarmos a lista, então foi criado um arquivo xml, com um texto simples, para que seja adaptado a lista.

Figura 15: Montando uma lista com os registros obtidos

```
if (tipo.equals("pais")) {
    c = bm.listarTodos("nome", "paises");// recebe os dados conforme os paramêtros solicitados
    List<Pais> listDados = new ArrayList<>();// list de dados
    while (c.moveToNext()) {
        Pais p = new Pais();
        p.setId(c.getInt(0));
        p.setNome(c.getString(1));
        p.setSigla(c.getString(2));
        listDados.add(p);
    } // percorre os dados do cursor, obtido na função listarTodos
    ArrayAdapter adapter = new ArrayAdapter(
            this, // contexto da listagem
            R.layout.layout_listagem_geral, // layout criado pra adaptação
            R.id.txtGeral, // widget (TextView) para exibir o conteúdo (criado dentro do layout de adaptação)
            listDados // array de dados
    );
    lista.setAdapter(adapter); // faz adaptação do layout criado
    c.close();
```

Fonte: Autor.

Após esta etapa, a lista está pronta para o usuário. A figura 16 mostra a lista produzida.

Figura 16: Imagem da lista.

Fonte: Autor.

Na aplicação foi desenvolvido um ContextMenu, para que os registros mostrados na lista possam ser editados. Para criar este menu, foi alterado na classe os métodos que trabalham com menu, que foram sobrescritos para controle das seções. Para a lista interagir com o menu é preciso adicionar à lista ao menu, com o método *registerForContextMenu(nome_da_lista)*, no construtor da classe. As figuras abaixo explicam o código produzido para formar o ContextMenu.

Figura 17: Criando opções do ContextMenu

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) {
    if (v.getId() == R.id.listGerDados) { // teste para buscar a lista correta
        menu.add("Editar Registro"); // adicionando itens ao menu
        menu.add("Excluir Registro");
        if (tipo.equals("locais")) {
            menu.add("Vizualizar no Mapa");
        }
        menu.add("Cancelar");
    }
}
```

Fonte: Autor.

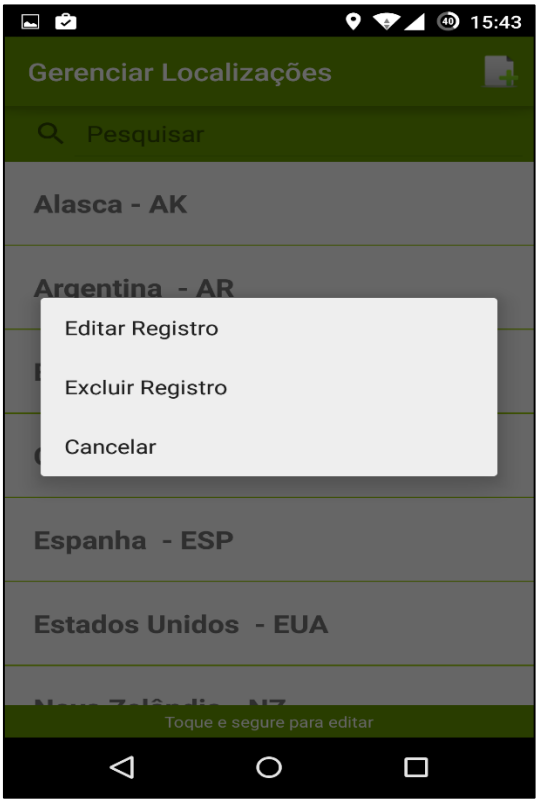
Figura 18: Busca de opções do ContextMenu

```
    AdapterView.AdapterContextMenuInfo acmi =
            (AdapterView.AdapterContextMenuInfo) item.getMenuInfo(); // pegando id itens da lista
    if (item.getTitle().equals("Editar Registro")) { // teste de opções do menu
        if (tipo.equals("pais")) {
            Pais pais = (Pais) lista.getItemAtPosition(acmi.position); // pegando objeto da lista
            Intent i = new Intent(getApplicationContext(), FormPaises.class); // Formulário
```

Fonte: Autor.

Após concluído as modificações o menu já está pronto, no caso do aplicativo, está disponibilizando opções de excluir registro, visualizar registro no mapa (caso a opção seja locais) e editar o registro levando a um formulário, formulário que é utilizado tanto para edição quanto para inserção de novos registros.

Figura 19: Imagem do ContextMenu



Fonte: Autor.

3.3.5 Formulários

No menu da tela de listagem (parte superior direito), temos um botão que leva o usuário a um formulário para cadastrar ou editar itens. O layout deste formulário foi criado com itens (*widgets*) semelhantes aos já apresentados no escopo deste projeto, os quais estão adicionados dentro de um *ScrollView* (barra de rolagem), caso o layout do formulário passe os limites do *display*.

Para a inserção de dados utilizamos um *ContentValues*, para definir os valores, e funções na classe que controla o banco de dados, para inserir estes valores. O código abaixo exemplifica a inserção ou alteração de valores:

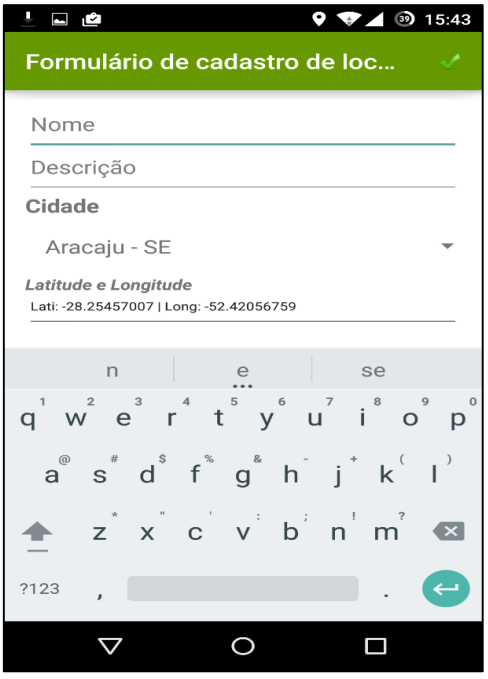
Figura 20: Inserindo registros no banco de dados

```
public void salvarCidade(){
    Estado e = (Estado) spinnerUF.getSelectedItem();
    values.put("nome", // nome do campo na tabela
            this.nomeCidade.getText().toString() // valor
    ); // variável values do tipo ContentValues
    values.put("estado", e.getId());
    if(opcao.equals("salvar")){
        bm.inserir("cidades", values); // tabela do banco e os valores
    }
    else{
        bm.atualizar(String.valueOf(id), "cidades",
                values);// tabela do banco e os valores e o id para atualizar neste caso
    }
    bm.close();
    finish();
}
```

Fonte: Autor.

O formulário de locais pega automaticamente os dados de latitude e longitude utilizando a mesma estratégia apresentada na tela inicial, apenas mudando os parâmetros do método *requestLocationUpdates* para que os dados de localização sejam atualizados com mais frequência. Os demais formulários seguem a mesma lógica de programação.

Figura 21: Imagem do formulário



Fonte: Autor.

3.3.6 Exibindo mapa e navegação

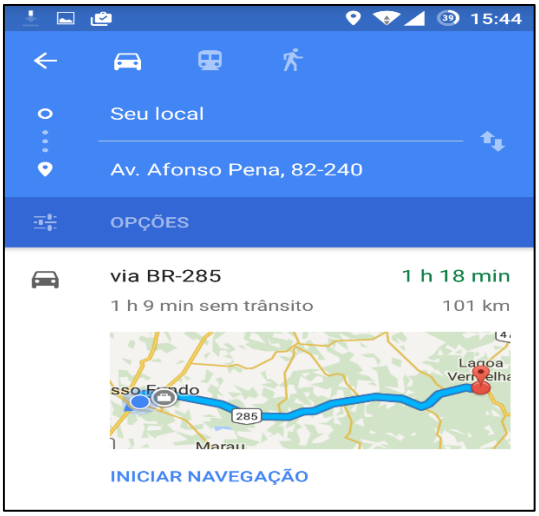
Quando o usuário salva um local desejado, ele aparecerá na lista de locais e, no *ContextMenu,* aparecerá uma opção para visualizar no mapa este ponto. Para produzir este mapa foi utilizado a mesma estratégia de programação da criação do mapa da tela inicial, qual mostra sua localização atual, a única diferença é que ao invés de pegar os dados de latitude e longitude do GPS ou da rede externa, está sendo pego dados salvos no banco de dados.

Figura 22: Imagem do mapa referente aos locais

Fonte: Autor.

Ao visualizarmos o local no mapa e clicarmos em cima do marcador renderizado ele nos oferece a opção de navegação, utilizando a navegação do *GoogleMaps*, que referencia seu ponto atual e traça uma rota até o ponto salvo (necessária conexão com a internet para navegar). A imagem abaixo demostra uma navegação a um ponto.

Figura 23: Imagem da tela de navegação



Fonte: Autor.

3.3.7 Banco de dados

Para que todos os dados da aplicação sejam adicionados ao banco de dados foi criada uma classe para auxiliar as transações com o banco, classe que estende a classe abstrata *SQLiteOpenHelper* que obriga a implementar 2 métodos abstratos que trabalham com a criação do bando de dados, o método *onCreate*, que faz a criação do banco de dados caso o banco não exista, e o método *onUpgrade*, que atualiza o banco de dados caso o mesmo já exista. Para as transações de dados (inserir, atualizar e remover) foram criados métodos para o controle das mesmas. A imagem abaixo exemplifica esta classe:

Figura 24: Classe do banco de dados (Criação do banco)

```
public class BancoManager extends SQLiteOpenHelper{

    private static final String NOME_BANCO = "localidade.db";
    private static int VERSAO = 1;

    public BancoManager(Context context) { super(context, NOME_BANCO, null, VERSAO); }

    @Override
    public void onCreate(SQLiteDatabase db) {...} // caso o banco não exista, cria um novo

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {...}
    // caso exista o banco, atualiza o banco de dados

    public void inserir(String tabela, ContentValues valores){
        this.getWritableDatabase().insert(tabela, null, valores);
    }

    public void atualizar(String id, String tabela, ContentValues valores){
        String [] argumentos = {id};
        this.getReadableDatabase().update(tabela, valores, "_id=?", argumentos);
    }

    public void remover(String id, String tabela){
        String [] argumentos = {id};
        this.getReadableDatabase().delete(tabela, "_id=?", argumentos);
    }
```

Fonte: Autor.

4. CONSIDERAÇÕES FINAIS

O objetivo principal do projeto foi alcançado, a produção de um aplicativo que gerencie pontos turísticos, salvando-os e mostrando-os em um mapa foi construído, junto a layouts que estão sendo utilizados em diversos aplicativos atuais.

Ainda serão necessários alguns ajustes na aplicação, para que a mesma fique mais elegante ao usuário, com mensagens interativas que auxiliem o usuário a utilizar a aplicação e alguns ajustes de desempenho e layout do aplicativo. Por mais que o aplicativo não esteja 100% completo, o seu propósito (salvar locais e mostra-los no mapa) está funcionando, junto a navegação do GoogleMaps.

A questão do usuário ter que cadastrar o país, estado e uma cidade, foi criada com o propósito de utilizar a aplicação em modo off-line, sem conexão com a internet, assim o usuário mesmo estando sem conexão poderá salvar o local desejado e visualiza-lo, a única função que estará impossibilitada de ser utilizada será a navegação, que necessita de uma rede externa para gerar uma rota de navegação. Tecnologias de busca automática de dados serão implementadas futuramente dentre outras funcionalidades que serão descritas no próximo tópico.

4.1 PROJETOS FUTUROS

Para um aplicativo mais completo e funcional, e também para que se possa gerar retornos lucrativos, é visado a implementação de tecnologias descritas abaixo:

- Implementação de cadastro automático, sem a necessidade do usuário digitar todos os dados de locais, utilizando um WEB Service do GoogleMaps com JSON para a busca de dados;
- Implementação de layouts mais interativos, facilitando a utilização do aplicativo;
- Implementação de comandos de voz, para que os dados possam ser salvos ou acessados de maneira eficaz ou que auxiliem, caso o usuário esteja dirigindo, por exemplo;
- Implementação de alertas (propagandas) para retorno financeiro do aplicativo produzido.

5. REFERÊNCIAS

ANDROID. *The Android History.* Disponível em: <http://www.android.com/history>. Acesso em: 05 nov. 2014.

ANDROID DEVELOPERS. *Android Location.* Disponível em: <http://developer.android.com/reference/android/location/package-summary.html>. Acesso em: 08 nov. 2014.

ANDROID DEVELOPERS. *Google Location Services API.* Disponível em: <http://developer.android.com/reference/com/google/android/gms/location/package-summary.html>. Acesso em: 08 nov. 2014.

ANDROID DEVELOPERS. *SQLite3.* Disponível em: < http://developer.android.com/tools/help/sqlite3.html>. Acesso em: 10 nov. 2014.

DOHERTY, Donald; MANNING, Michelle. Borland JBuilder 2: *in 21 days.* Indiana: Sams, 1998.

GOOGLE DEVELOPERS. *API Android do Google Maps - Biblioteca externa.* Disponível em: < https://developers.google.com/maps/documentation/android/?hl=pt-br>. Acesso em: 09 nov. 2014.

GOOGLE DEVELOPERS. *Google Maps Android API.* Disponível em: <https://developers.google.com/maps/documentation/android/intro?hl=pt-br>. Acesso em: 10 maio. 2015.

GOOGLE PLAY. Google Play Services. Disponível em: <https://play.google.com/store/apps/details?id=com.google.android.gms&hl=pt_BR>. Acesso em: 07 nov. 2014.

IDG News Service. *Android atinge 85% de participação no mercado mundial de smartphones*, 2014. Disponível em: <http://idgnow.com.br/mobilidade/2014/07/31/android-atinge-85-de-participacao-no-mercado-mundial-de-smartphones/>. Acesso em: 03 nov. 2014.

KARASINSKI. Eduardo. *O que é geolocalização*. Disponível em: <http://www.tecmundo.com.br/o-que-e/3659-o-que-e-geolocalizacao-.html>. Acesso em: 06 nov. 2014.

OPENSOFT. *Introdução ao Java.* Disponível em: <http://www.opensoft.com.br/siteosi/jsp/JavaBasico.jsp>. Acesso em: 03 nov. 2014.

ORACLE. *Oracle and Sun Microsystems*. Disponível em: <http://www.oracle.com/us/sun/index.html>. Acesso em: 05 nov. 2014.

PEREIRA, Fernando. Programação para a plataforma Android: *Localização.* Disponível em: <http://homepages.dcc.ufmg.br/~fernando/classes/android/slides/Class13.pdf>. Acesso em: 07 nov. 2014.

SQLITE. *About SQLite*. Disponível em: < http://www.sqlite.org/about.html>. Acesso em: 10 nov. 2014.

SOUZA, Felipe A. Gavazza. *Tecnologia ART: o que realmente é e quais as suas vantagens,* 2014. Disponível em: <http://www.androidpit.com.br/tecnologia-art-dalvik >. Acesso em: 02 nov. 2014.

TECHTUDO. *Google Maps: faça download no celular e chegue logo ao seu destino.* Disponível em: < http://www.techtudo.com.br/tudo-sobre/google-maps.html>. Acesso em: 09 nov. 2014.

TAVARES, Alexandre. O que é Google Play. Disponível em: <http://smartmundo.com/o-que-e-play-store/>. Acesso em: 08 nov. 2014.

UOL Economia. *Brasil deve se tornar o quarto maior consumidor de smartphones até 2016,* 2012. Disponível em: <http://economia.uol.com.br/ultimas-noticias/infomoney/2012/03/20/brasil-deve-se-tornar-o-quarto-maior-consumidor-de-smartphones-ate-2016.jhtm>. Acesso em: 03 nov. 2014.