UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

FRANCISCO WITALO BENICIO DA SILVA

EASYADVOGADOS, UM APLICATIVO BASEADO EM GEORREFERENCIAMENTO PARA FACILITAR A BUSCA POR ADVOGADOS

QUIXADÁ
2014

FRANCISCO WI TALO BENICIO DA SILVA

EASYADVOGADOS, UM APLICATIVO BASEADO EM GEORREFERENCIAMENTO PARA FACILITAR A BUSCA POR ADVOGADOS

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Sistemas de Informação/Bacharelado da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Orientador Prof. MSc Regis Pires Magalhães

QUIXADÁ
2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

S578e Silva, Francisco Witalo Benicio da
EasyAdvogado: um aplicativo baseado em georreferenciamento para facilitar a busca por advogados / Francisco Witalo Benicio da Silva. – 2014.
70 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2014.
Orientação: Prof. Me. Regis Pires Magalhães
Área de concentração: Computação

1. Dispositivos móveis 2. Programa de computador 3. Software - Desenvolvimento I. Título.

CDD 005.1

FRANCISCO WITALO BENICIO DA SILVA

EASYADVOGADOS, UM APLICATIVO BASEADO EM GEORREFERENCIAMENTO PARA FACILITAR A BUSCA POR ADVOGADOS

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Aprovado em: _____ / novembro / 2014.

BANCA EXAMINADORA

_____
Prof. MSs. Regis Pires Magalhães (Orientador)
Universidade Federal do Ceará-UFC

_____
Prof. MSs. Bruno Gois
Universidade Federal do Ceará-UFC

_____
Prof. MSs. Márcio Espíndola Freire Maia
Universidade Federal do Ceará-UFC

AGRADECIMENTOS

À minha mãe, que desde o começo, mesmo antes de entrar na faculdade, me deu suporte em toda minha vida.

À minha namorada, que sempre me ajudou e me apoiou quando mais precisei.

Aos meus amigos de graduação, que sempre ajudaram uns aos outros, que compartilharam os seus conhecimentos com os outros e que ajudaram uns aos outros a chegar até o fim.

À todos os professores, de maneira geral, pelas contribuições na minha vida acadêmica e profissional.

Ao professor Regis pelas contribuições dadas à esse trabalho.

Ao professor Regis, pela forma de ensino totalmente prática ministrada na cadeira de Desenvolvimento Web, fazendo com que não somente eu, mas todos os alunos ficassem mais motivados em aprender.

Ao professor Márcio, que sempre acreditando em mim, me deu minha primeira oportunidade de trabalho, assumindo reais responsabilidades e me permitindo construir em mim mesmo um bom profissional.

Ao professor Wladimir, que dedicou seu tempo para me ajudar e me motivar a não desistir quando estava com dificuldades e sem foco na faculdade.

Um sincero, muito Obrigado.

"The best way to predict your future, is to create it."
(Abraham Lincoln)

RESUMO

Dispositivos móveis têm se tornado ferramentas importantes para as mais diversas áreas de atuação, devido ao crescente número de usuários, como também aos recursos oferecidos por este tipo de tecnologia. Um desses recursos, é o de georreferenciamento, que permite o uso da localização do usuário para oferecer diferentes serviços com base nesta informação. Desse modo, uma área que faz bom uso desse recurso, é a de publicidade baseada em georreferenciamento, possibilitando ao usuário receber informações com base na sua posição geográfica. O EasyAdvogados faz uso do conceito de georreferenciamento, para fornecer informações sobre vários advogados espalhados no mapa, baseado na posição do usuário. Sendo assim, quem fizer uso do aplicativo pode saber quais advogados estão por perto dele, separá-los por categoria (área de atuação), além de poder telefonar, enviar email e ver fotos de um determinado advogado. O aplicativo também facilita a visualização de processos de primeira instância, possibilitando ao usuário ver o seu processo, economizando tempo e esforço de entrar em contato com o seu advogado para saber sobre seu processo. O EasyAdvogados surgiu com a necessidade dos advogados divulgarem sua atuação, pois devido a lei nº 8.906, de 4 de julho de 1994, eles são impossibilitados de fazerem qualquer tipo de publicidade convencional. Também surgiu da necessidade de prover uma ferramenta simples e útil para que qualquer usuário de aplicativos móveis possa realizar a busca por advogados. Muitas vezes um usuário sente a necessidade do serviço de advocacia, porém não conhece advogados dessa área e acaba não sabendo a quem recorrer ou pedindo a indicação de advogados para a OAB, que não pode fornecer indicações de advogados por lei. Unindo essas duas necessidades, surge o EasyAdvogados. Aproveitando o grande número de usuários de dispositivos móveis e a possibilidade de utilizar informação baseada em contexto (localização), o EasyAdvogados se torna uma ferramenta de publicidade para os advogados e uma ferramenta eficiente de busca de advogados para os usuários. Sendo assim, o aplicativo favorece dois tipos de público-alvo: advogados que precisam se tornar mais vistos no mercado e clientes que precisam de advogados em áreas específicas de maneira fácil e rápida.

Palavras-chave: Desenvolvimento Móvel, Georreferenciamento, Publicidade na Advocacia, Busca de Advogado.

ABSTRACT

Mobile devices have been proven important tools for several areas, due to the increasing number of users that adapt this kind of technology, as well as the resources provided by these platforms. One of these resources, is called the georeferencing, which gets the user position to offer the most different services based on this information. Thus, an area that makes good use of this resource is the advertising based on georeferencing. It enables the user to receive information based on his geographic position. *Easy Advogados* uses the concept of georeferencing to give information about the several lawyers distributed in the map based on user position. Therefore, the app user can know which lawyers are near, filter lawyers by category (field), call, send email and see photos of a specific lawyer. The application also facilitates the visualization of first instance processes, giving the user the possibility to see the process directly in the app, saving time and effort to call his lawyer in order to know details about his process. The app started with the need of the lawyers to propagate information about their jobs, because due to the law No. 8.906, July 4 of 1994 from the Code of Ethics of Brazillian Lawyers, they are unable to do any type of conventional advertising. It also emerged from the needs to provide a simple and useful tool to help users search for lawyers. Many times an user feels the need to use a service provided by a lawyer, but doesn't know any lawyers in a specific field and ends not knowing who to ask or even requesting an indication to Order of Brazilian Lawyers that can't provide lawyes indications by law. The Easy Advogados app was created to join these two needs. Availing the big number of mobile devices users and the possibility of use this context based information (localization), the Easy Advogados app became a advertising tool for the lawyers and an efficient lawyer search tool to the users. Thus, the app has two main of audiences: lawyers that want to become more visible in the market and clients that need lawyers in specific areas in a fast and easy way.

Keywords: Mobile Development, Georeferencing, Advocacy Advertising, Search for a Lawyer.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

Nos dias de hoje, os dispositivos móveis (e.g. *smartphones* e *tablets*) vêm sendo cada vez mais utilizados pela população para as mais diversas finalidades (e.g., diversão, negócios e turismo). Esse fato contribui não só para que a sociedade se torne mais ativamente conectada, mas também possibilita o surgimento de ferramentas que auxiliam e facilitam a vida da população conectada. Esse ambiente de informações e conexões, torna possível a criação de ambientes ubíquos, onde as pessoas podem ter acesso a informação em todo lugar e a qualquer momento.

Um fator importante a ser destacado, é o uso cada vez maior de aplicações georreferenciadas (baseadas em localização), seja na área social, marketing, jogos e aplicações diversas. A disponibilização desse recurso pelas plataformas móveis, possibilita aos desenvolvedores a construção de aplicações que fornecem diferentes tipos de informações baseadas na localização do usuário, trazendo assim informações que podem ser mais úteis para o usúario.

Fazendo uso do serviço de georreferenciamento disponibilizado pela plataforma Android, o aplicativo Easy Advogados traz informações sobre os diversos advogados existentes de maneira clara e fácil para o usuário do aplicativo. Baseado na localização, é fornecida uma lista de advogados próximos e suas localizações marcadas no mapa. Além do poder da busca por localização, também é possível filtrar a busca por área de atuação e por nome.

O aplicativo surgiu de duas necessidades. 1) Muitas pessoas precisam de advogados, porém muitas vezes não conhecem ou não têm indicação de um, ou precisam de um tipo de advogado para uma área específica. 2) Advogados praticamente não utilizam publicidade ao seu favor, pois a propaganda na advocacia é apenas de caráter informativo, tendo como meta mostrar a sociedade em geral, ou a uma clientela já existente, dados e informações objetivos acerca da atividade realizada pelo advogado ou sociedade de advogados, bem como suas características, área de atuação e qualificações (v. art. 1º e 2º, do Provimento OAB nº 94/2000, bem como arts. 5º e 28, do Código de Ética e Disciplina).

Deste modo, o Easy Advogados se torna uma ótima opção para os advogados fazerem propaganda de maneira mais abrangente e contextual para conseguirem novos clientes, não somente fornecendo informações a clientes já existentes mas, principalmente, possibilitando aos usuários do aplicativo, uma ferramenta ágil, prática e fácil na busca por advogados.

A organização deste trabalho obedece a seguinte sequência: A seção 2 apresenta os conceitos utilizados neste trabalho. A seção 3 apresenta os trabalhos relacionados a este, mostrando no que se assemelham e se diferem. A seção 4 explica os passos do processo de desenvolvimento do aplicativo. Na seção 5 serão explicados os passos de desenvolvimento, detalhando como os passos foram executados. A seção 6 mostra o resultado do trabalho e, finalmente, a seção 7 trata das considerações finais e dos trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Aqui serão apresentados e explicados alguns conceitos utilizados durante a elaboração deste trabalho.

2.1. Android

Android é um sistema operacional para dispositivos móveis baseado no Linux e seu desenvolvimento foi liderado pelo *Google*[1]. Foi primordialmente desenvolvido para *smartphones* e *tablets*, mas atualmente já está entrando em uso por TVs, carros e *smart watches*. O Android utiliza como entrada métodos de toque, *swipe*, *pinching*, etc.

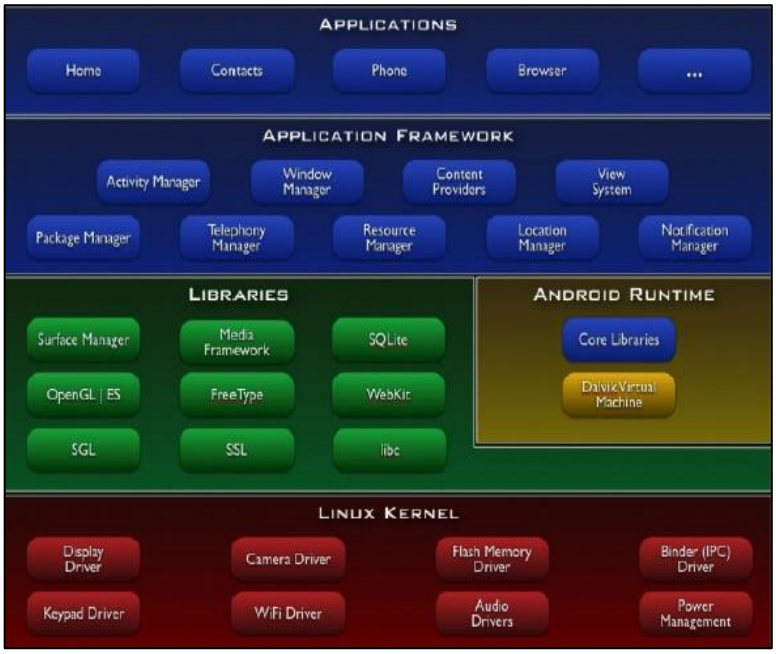
De acordo com (GOOGLE, 2014) é o Sistema Operacional móvel mais instalado mundialmente e está presente em mais de 190 países. Todo dia mais de 1 milhão de novos Androids são utilizados pela primeira vez. Segundo a Strategy Analytics[2], o sistema operacional Android, na metade de 2013, já possuia cerca de 85% do mercado, enquanto os concorrentes principais, iOS e Windows Phone, ocupavam respectivamente 12% e 3%. Em 2014 o Android diminuiu seu mercado em 1%, mas mesmo assim, ainda domina uma enorme fatia de mercado com 84%.

2.1.1. Arquitetura da plataforma Android

A arquitetura da plataforma Android é formada por várias camadas (GOOGLE, 2014) como é possível ver na Figura 1, também chamadas de *pilhas de software*, onde cada camada possui sua função específica.

[1] https://www.google.com.br/intl/pt-BR/about/company/
[2] http://www.strategyanalytics.com

Figura 1 - Arquitetura Android



Fonte: Google, 2014.

2.1.1.1. Aplications

No topo da pilha, encontram-se diversas aplicações disponíveis pelo sistema tais como: email, calendário, relógio, aplicações de terceiros e até mesmo funções mais básicas como ligações telefônicas. A interação com os mais diferentes aplicativos ocorre nessa camada, enquanto as demais ficam acessíveis apenas a desenvolvedores e fabricantes de software.

2.1.1.2. Application Framework

A camada Application Framework fica responsável pelos programas que gerenciam as funções básicas do telefone, como alocação de recursos, aplicações do telefone, mudança entre processos, etc. Possibilita que os desenvolvedores possam ter acesso total ao sistema, tirando vantagens das capacidades do hardware, processamento em background, alertas, serviços, etc. Também simplifica o desenvolvimento e reuso, além de possibilitar a construção de aplicações mais complexas a partir de outras mais simples.

2.1.1.3. Libraries

A camada *Libraries* disponibiliza várias bibliotecas que possibilitam acesso aos componentes do sistema, como, por exemplo, banco de dados entre outros.

2.1.1.4. Android Runtime

Um conjunto de bibliotecas do núcleo Java são inclusas na camada Android Runtime, e é nela também que se encontra a maquina virtual Dalvik, na qual são executados os processos de aplicação de uma forma otimizada para o baixo consumo de memória. Os arquivos Dalvik executable são resultantes da compilação.

2.1.1.5. Linux Kernel

A base da pilha é o *kernel* do Android baseado na versão 2.6 do Linux. Inclui os programas de gerenciamento de memória, configurações de segurança, gerenciamento de energia, vários *drivers* e *hardware*.

2.1.2. Componentes do Android

Elementos básicos de aplicações podem ser compartilhados no Android e se forem concedidas permissões a esses elementos é possível usar recursos já desenvolvidos em outras aplicações. Com o Android, processos de aplicação podem ser iniciados somente quando necessários, assim como permitir a instanciação de objetos Java. A maioria dos sistemas e aplicações têm um único ponto de entrada, ou função principal. Aplicações Android invocam componentes centrais que são instanciados e executados quando eles forem necessários. Existem quatro tipos principais de componentes Android: *activities, services, broadcat receivers* e *content providers*.

Activities são componentes utilizados para controlar o comportamento do *layout* de uma tela, ou seja, as diferentes respostas para as diferentes ações recebidas. Os *services* são utilizados para realizar tarefas em *background* e por tempo indeterminado, deste modo, não possuem interface visual. *BroadCast Receivers* são componentes simples que recebem e reagem à transmissão de anúncios (intenções). Os *Content Providers* são responsáveis pela

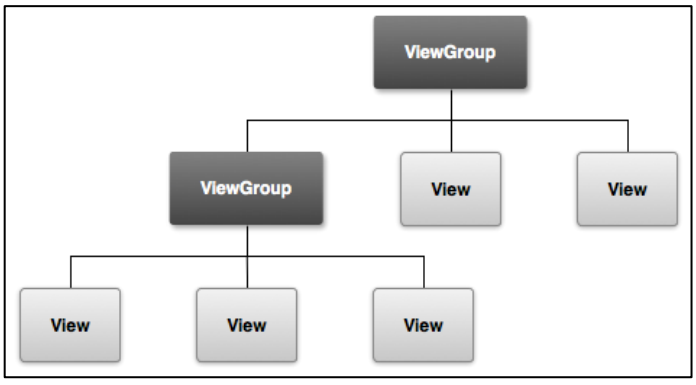
persistência de dados, como por exemplo o SQLite, fazendo com que tais dados possam ser acessados por aplicações e *activities*.

2.1.3. Interface do Usuário

A interface de usuário de uma aplicação Android é tudo o que o usuário pode ver e interagir. A plataforma dispõe de uma série de componentes de interface pré-implementados como *layouts* e controles de interface que possibilitam ao desenvolvedor construir a interface gráfica do aplicativo. Além desses componentes de composição de telas, também estão disponíveis interfaces especiais usadas para mostrar as mais diferentes informações como os *dialogs, notifications e menus*.

Todos os elementos de interface do Android são construídos a partir de uma *View* ou de um *ViewGroup*. Uma *view* é um objeto que desenha algo na tela e que o usuário pode interagir. Já um objeto *viewgroup* contém outros objetos *view* e, também, *viewgroup.* Toda a tela de uma aplicação é uma combinação hierárquica de componentes *ViewGroup* e *View*. Os objetos *ViewGroup* funcionam como organizadores, separando os diferentes componentes na tela. As *Views* podem servir como métodos de entrada ou saída, ou componentes visuais. Na Figura 2 podemos observar um exemplo dessa hierarquia.

Figura 2 - Exemplo de hierarquia de componentes



Fonte: Google, 2014

No Android existem os mais variados tipos de objetos de composição de *layout*, todos eles estendendo o objeto *View*. Para usá-los pode-se instanciá-los via código e adicionar programaticamente, ou então, de uma maneira mais prática, utilizar elementos de XML que podem ser definidos nos arquivos de *layout* e referenciados via código.

Os componentes principais de uma interface do Android são: os *layouts* (*LinearLayout, RelativeLayout, GridView* e *ListView*), os controles de entrada (*Buttons, EditTexts, RadioButton, CheckBoxes, Spinners* e *Pickers*), os componentes para exibir informações (*TextView e ImageView*), a *actionbar*, menus, componentes de configuração de aplicação, entre outros. Esses componentes são descritos a seguir.

- ***LinearLayout*** - utilizado para organizar *Views* verticalmente ou horizontalmente;
- ***RelativeLayout*** - utilizado para organizar *Views* de maneira relativa à tela;
- ***GridView*** - usado para mostrar itens em um *grid*, tendo opções de número de colunas, espaço horizontal e vertical, etc;
- ***ListView*** - componente utilizado para mostrar itens em uma lista vertical;
- ***Button*** - componente utilizado para servir como um botão, respondendo a cliques do usuário;
- ***TextField*** - também chamado de *EditText*, é o componente onde o usuário pode digitar textos, números, etc;
- ***RadioButton*** - usado para que o usuário escolha uma única opção;
- ***CheckBox*** - semelhante ao *RadioButton*, mas geralmente usado para opções booleanas ou de múltipla escolha;
- ***Spinner*** - exibe uma lista de opções para que o usuário possa escolher;
- ***TextView*** - componente utilizado para mostrar um texto na tela;
- ***ImageView*** - utilizado para exibição de imagens;
- ***ActionBar*** - componente atualmente suportado por todas as versões do Android a partir da 2.2, que contém algumas ações que o usuário pode utilizar, de acordo com o desenvolvimento, bem como segurar *menus* para aplicação;
- ***Menus*** - existem diferentes tipos de menus, mas atualmente, o mais recomendado, para a estrutura específica deste projeto, segundo (GOOGLE, 2014) é o *NavigationDrawer,* que exibe um menu lateral para o usuário que pode ser mostrado ao arrastar o dedo na tela da esquerda para direita (ou direita para a esquerda) e tem como *layout* um *fragment.*
- ***Settings*** - utilizado para construir a tela de configurações da aplicação.

2.1.4. Activities

De acordo com (LECHETA, 2013) uma *activity* é o componente do Android responsável por representar uma interface visual, ou seja, com este componente é possível controlar o comportamento do *layout* da aplicação. Dentro de uma aplicação podemos ter somente uma *activity* controlando somente uma tela durante toda a aplicação, bem como uma *activity* controlando vários *fragments* com várias telas, várias *activities* com várias telas e várias *activities* com vários *fragments* e várias telas.

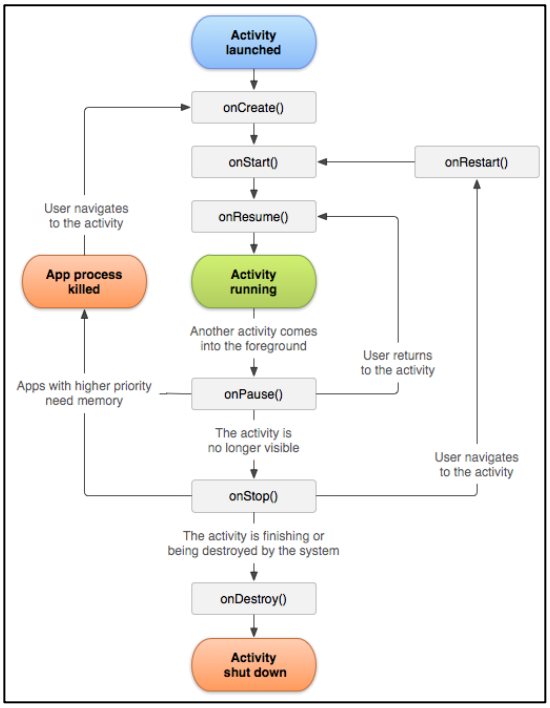
Como método principal da *activity* temos o método *onCreate()* que é chamado logo que a mesma inicia. Nesse método o desenvolvedor passará o *layout* daquela *activity*, bem como iniciará os componentes da tela (*ImageView, TextView, EditText, etc).*

Uma *activity* possui um ciclo de vida que pode ser usado em diferentes momentos e que é de grande utilidade para os desenvolvedores, pois assim, podem controlar o comportamento de uma *activity* nos seus diferentes estados.

2.1.4.1. Ciclo de Vida

As fases do ciclo de vida de uma activity podem ser vistas na Figura 3 e são chamadas em diferentes momentos. A seguir será descrito quando cada método é chamado na *Activity*.

Figura 3 - Cliclo de vida de uma Activity



Fonte: Google, 2014

- ***onCreate()*** - Chamado quando a *activity* é criada. Nesse método as instâncias principais devem ser criadas, como por exemplo, instanciação de *views*, passagem de dados para uma lista, etc. Sempre seguido pelo método onStart();
- ***onStart()*** - Chamado antes da *activity* ficar visível para o usuário. Seguido do método onResume(), caso a tela fique visível, ou onStop() caso fique escondida;
- ***onRestart()*** - Chamado logo após a *activity* parar, imediatamente antes de ser chamada novamente;
- ***onResume()*** - Chamado antes da *activity* começar a interação com o usuário. Aqui a *activity* já está no topo da pilha de atividades. Sempre seguido do método *onPause();*
- ***onPause()*** - Chamado quando o sistema chama uma outra *activity* para assumir a frente da tela. Esse método geralmente é utilizado para parar componentes que possam consumir muita memória na aplicação. Seguido pelo *onResume()* ou *onStop();*
- ***onStop()*** - Chamado quando a *activity* não está mais visível para o usuário, podendo acontecer por ser destruída ou então outra *activity* já está sendo mostrada para o usuário. Seguido pelo *onRestart()* ou pelo *onDestroy();*
- ***onDestroy*** - Chamado pouco antes da *activity* ser destruída. Esse é o último método a ser chamado na *activity.*

2.1.5. Fragments

Segundo (LECHETA, 2013), *Fragments* representam um comportamento específico ou uma parte de uma *activity.* Podem ser reusados em diferentes *activities*, podem ser combinados em uma única *activity* para criar um painel *Multi-Pane*. Pode-se pensar em um *fragment* como uma seção modular de uma *activity* que tem seu próprio ciclo de vida, recebe suas próprias entradas e que pode ser removido ou inserido enquanto a *activity* está sendo executada.

Um *fragment* sempre estará ligado a uma *activity*, que não necessariamente será a mesma, e o seu ciclo de vida é afetado diretamente pelas mudanças ocorridas no ciclo de vida da *activity* ao qual está ligado. Por exemplo, se a *activity* for destruída o *fragment* também será e do mesmo modo se for parada, mas caso a ela esteja em execução, é possível manipular os *fragments* de forma a removê-los ou adicioná-los.

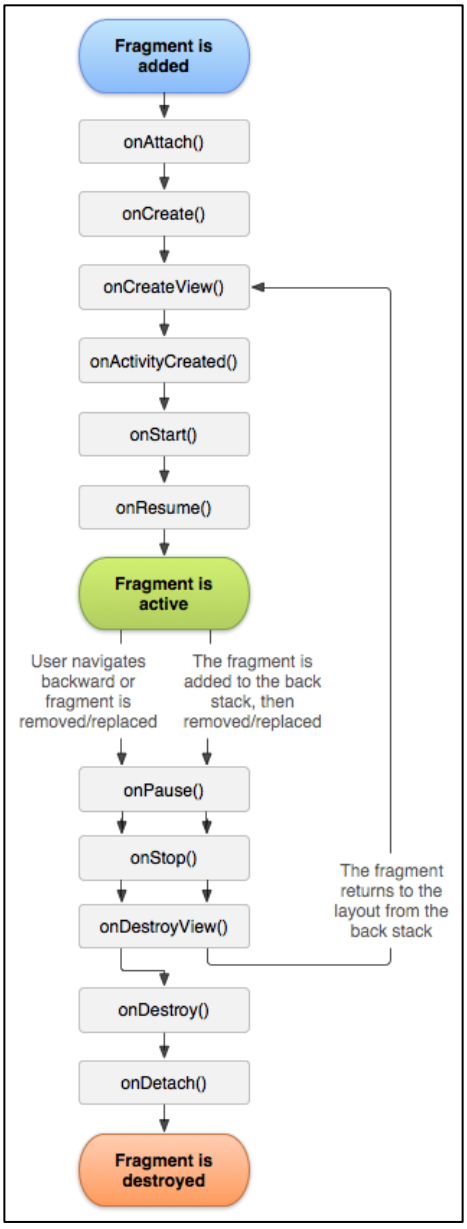
O *fragment*, que possui seu próprio layout, se encontra dentro de um *ViewGroup* presente no *layout* da *activity* responsável por ele. Um *fragment* pode ser adicionado a uma

activity a partir do *layout* adicionado a tag <*fragment*> ou então programaticamente, adicionado a um *ViewGroup* presente no *layout* da *activity*.

2.1.5.1. Ciclo de Vida

O ciclo de vida de um *fragment* é bem semelhante ao de uma *activity*. Desse modo, a seguir serão explicados somente os três principais métodos a serem implementados quando se cria um *fragment.*

Figura 4 - Ciclo de vida de um Fragment



Fonte: Google, 2014.

- ***onCreate()*** - Chamado quando o *fragment* está sendo criado. Na implementação deste método, os principais componentes devem ser inicializados;
- ***onCreateView()*** - Chamdo quando o *fragment* irá desenhar na tela a interface de interação com o usuário pela primeira vez. Para que isso seja possível é necessário que neste método seja retornada uma *View* que é o próprio *layout* implementado, para ser mostrado;
- ***onPause()*** - Este método é chamado logo que o sistema percebe que o *fragment* sairá de vista (o que não significa que será destruído). Aqui é onde geralmente mudanças ocorridas devem ser salvas ou persistidas;

2.1.6. Intents

Intents são um dos principais recursos presentes no Android. Eles atuam como componentes de mensagens, utilizados para requisitar uma ação de um outro componente de aplicação, seja este componente da sua própria aplicação ou de outras aplicações. Intents funcionam como facilitadores de comunicação entre componentes e possuem três tipos principais de uso:

- **Executar uma *activity*** - Para que se possa executar uma instância de uma *activity*, deve-se passar um *Intent* para o método *startActivity()*. O *intent* irá ficar responsável por descrever a *activity* a ser executada e ainda passará dados de uma *activity* para a outra.
- **Executar um *service*** - Um *service* é um componente que executa operações em background, sem necessidade de uma interface de usuário. Para que se possa executar um *service* é necessário passar um *intent* para o método *startService()*;
- **Entregando resultados em *Broadcast*** - Um *broadcast* é uma mensagem que qualquer aplicativo pode receber. O sistema entrega diferentes mensagens em broadcasts relativos a eventos do sistema. Para que se possa entregar tais resultados para um aplicativo, deve-se passar um *intent* para o método *sendBroadcast()*;

2.1.6.1. Tipos de Intents

Existem dois tipos de *Intents*, os explícitos e os implícitos:

- **Explícitos** - explicitamente é passado o nome do componente a ser executado. Geralmente *intents* explícitos são utilizados para executar um componente do próprio aplicativo.
- **Implícitos** - uma ação é declarada para o sistema, e este fica responsável por encontrar os aplicativos responsáveis por lidar com esta ação.

2.1.7. Opções de Armazenamento

De acordo com (GOOGLE, 2014) o Android provê uma série de opções para que seja possível salvar dados localmente. De acordo com a necessidade específica, o desenvolvedor pode escolher como esses dados podem ser salvos. Dependendo do tipo de dado a ser salvo, se deve ser privado, se deve ser acessível por outros componentes e o quanto de espaço necessita para ser salvo.

Existem cinco opções disponíveis para uso:

- ***Shared Preferences*** - Salva dados primitivos (*Strings*, inteiros, *booleans*, *double*, etc) em pares de chave e valor;
- **Armazenamento Interno** - Salva dados primitivos na memória do dispostivo;
- **Armazenamento Externo** - Salva dados primitivos em um armazenamento externo compartilhado;
- ***SQLite*** - Salva dados estruturados em um banco de dados relacional local e privado;
- **Conexão com a Internet** - Salva dados estruturados na Web no servidor privado do desenvolvedor.

A seguir serão explicados os dois tipos de armazenamento utilizados neste trabalho pelo aplicativo desenvolvido.

2.1.7.1. Shared Preferences

A classe *Shared Preferences* provê meios para que dados possam ser salvos e, posteriormente, utilizados como chave e valor. Todos os valores primitivos (boolean, String, int, double, etc) podem ser salvos na classe *Shared Preferences*, e são persistidos passando

por diferentes sessões de usuário, podendo ser utilizados posteriormente, mesmo que a aplicação seja destruída.

Para que se possa utilizar esta classe, dois tipos de métodos estão disponíveis:

. *getSharedPreferences()* - usado quando se tem a necessidade de diferentes arquivos de preferência identificados por nome;

. *getPreferences()* - utilizado quando se faz uso das preferências de uma *activity* em particular.

Para salvar valores, é necessário chamar o método *edit()* para que se possa receber um objeto do tipo *SharedPreferences.Editor*. Após receber o objeto, métodos como *putBoolean(), putString()*, etc, podem ser utilizados para persistir dados. Para ler valores, usa-se os métodos *getBoolean e getString()*, por exemplo.

2.1.7.2. SQLite

O Android dá suporte a todos os recursos existentes em um banco de dados relacional local SQLite. Qualquer banco criado será acessível, em qualquer classe, por nome, porém não será acessível fora da aplicação.

O método recomendado por (GOOGLE, 2014) para se criar um banco de dados SQLite no Android, é criar uma subclasse da classe *SQLiteOpenHelper* e sobrescrever o método *onCreate()*, onde o comando de criar tabelas no banco de dados pode ser executado. Um exemplo pode ser visto na Figura 5.

Figura 5 - Exemplo de uso do AQLite no Android

```
public class DictionaryExample extends SQLiteOpenHelper {
    private static final int DATABASE_VERSION = 1;
    private static final String DICTIONARY_TABLE_NAME = "dicionário";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";
    DictionaryExample(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

Fonte: próprio autor.

Com relação a consultas no banco, várias podem ser feitas utilizando os métodos de *query()* do SQLiteOpenHelper. Esses métodos aceitam diferentes parâmetros, tais como a tabela a ser buscada, projeções, seleções, colunas, agrupamento, entre outros. Para consultas mais complexas, de acordo com (GOOGLE, 2014), é recomendável utilizar o *SQLiteQueryBuilder* que provê diversos métodos para construção de consultas.

2.2. Location Based Services (LBS)

Aqui serão apresentados alguns conceitos sobre Serviços Baseados em Localização, assunto que serve como uma das bases deste trabalho. Algumas definições de LBS serão apresentadas, bem como as tecnologias que podem ser utilizadas e, por fim, como LBS podem ser usados neste e em diferentes projetos.

2.2.1. Definição de LBS

Não existe uma definição exata do que seriam LBS, visto que dependendo da área de enfoque, podemos ter diferentes conceitos para este mesmo assunto.

Segundo (KUPPER, 2005) uma definição apresentada pela GSM Association é que serviços baseados em localização utilizam um "alvo" para agregar valor ao serviço provido, onde o alvo seria, por exemplo, a posição de um usuário. Ainda como valores adicionais a serem agregados neste tipo de serviço, seria o fato de poder mostrar esta posição em um mapa ou até mesmo o acionamento automático de um serviço quando uma posição for atingida.

Ainda de acordo com (KUPPER, 2005) existe um outro grupo de serviços chamado de *context-aware services*, em que os LBS estão inseridos. Estes serviços, além de fazerem uso da informação geoespacial, utilizam informações sobre contexto e costumes de usuários para fornecerem informações relativas.

Outra definição apresentada por (FERRARO, 2010) é que serviços baseados em localização podem ser definidos como serviços provedores de informações, disponíveis em dispositivos móveis, através do uso de redes móveis, que têm a habilidade de fazer uso da

posição geográfica de um usuário para um fim específico. Em resumo, o autor propõe três questões que são apresentadas por um LBS:

. "Onde estou?" - Referente a posição;
. "O que posso fazer por perto?" - Informações relacionadas a posição;
. "O que eu acho desse lugar" - Interação e contexto referente a posição.

2.2.2. Tipos de Serviço

Segundo (CHEN, 2005) serviços baseados em localização podem ser de três tipos:

. ***Push*** - O próprio serviço percebe a mudança ou obtenção de localização do usuário e oferece uma resposta para tal;
. ***Pull*** - O usuário requisita uma informação do serviço e informa sua localização para o mesmo;
. ***Poll*** - Requisições periódicas são feitas com base em um protocolo implementado.

2.2.3. Tecnologias de Localização

Existem diversas tecnologias para a obtenção de localização de um ponto geográfico específico. Elas diferem pela velocidade de captação, precisão, área de abrangência, entre outros.

A seguir temos uma breve explicação das possíveis tecnologias utilizadas em dispostivos móveis:

. **GPS** - *Global Positioning System* é uma tecnologia desenvolvida em 1973 pelo exército dos Estados Unidos da América projetada para fornecer a localização instantânea e velocidade de deslocamento de um ponto sobre um mapa. O sistema funciona entre uma transmissão via rádio de vários satélites para um receptor da sua posição geográfica. Atualmente as tecnologias de posicionamento desse tipo incluem não somente o GPS, mas também aGPS, AGPS e GLONASS, em que este último é um sistema de posicionamento desenvolvido pelos russos, que está presente na maioria dos *smartphones* novos e é considerado mais preciso do que o GPS.

. **Cell ID** - É utilizada, por exemplo, em redes GSM, atuando da seguinte maneira: utilizam-se estações base (antenas) e potência do sinal, para assim saber a distância entre o usuário e a fonte.

. **Wifi** - A IEE-802.11, mais conhecida como WiFi (Wireless Fidelity) pode fornecer o local de um usuário com base na localização do *access-point* e força do sinal.

2.2.4. Localização no Android

De acordo com (GOOGLE, 2014) a plataforma Android dispõe de recursos para acessar o serviço de obtenção de localização do dispositivo. Este recurso está presente na API *android.location*. Para tal, pode-se registrar um *listener*, que é uma classe responsável por receber as variações de posição do usuário ou alertas de proximidade.

A classe *LocationManager* é responsável por prover o acesso ao serviço de localização. Já a classe *LocationListener* pode ser registrada juntamente com o *LocationManager* para receber as variações de posição do usuário de acordo com um tempo ou deslocamento. A classe *LocationProvider* é responsável por determinar a tecnologia a ser utilizada para a obtenção de posição, podendo oferecer o provedor GPS ou Network (GPS, WiFi, GSM).

Além do serviço de localização por latitude e longitude, o Android também dispõe de um serviço de localização baseado em endereço, ou seja, dá a capacidade de procurar a localização do usuário com base na rua, bairro, etc. Essa classe é chamada *Geocoder* e é acessada através de uma requisição via Internet.

2.2.5. GoogleMaps

De forma a se fazer uso da localização do usuário, a exibição desta localização e de outras próximas em um mapa, se torna um fator interessante a ser explorado.

Para isso, de acordo com (GOOGLE, 2014) isso é possível no Android, graças a GoogleMaps API V2, que permite utilizar um mapa do Google no Android, marcar pontos, definir tamanho, zoom, animações, etc.

De modo que se possa fazer uso do mapa em uma aplicação Android, é necessário registrar o pacote desta app no console do Google[3]. Após esse registro, deve-se informar as

[3] www.console.developers.google.com

APIs que essa aplicação poderá utilizar e assim gerar uma *MapsAPIKey* que dá acesso ao uso em si dos mapas.

Para inserir um mapa em um layout do Android, é preciso criar um *fragment* e passar como seu *name*, o valor "*com.google.android.gms.maps.SupportMapFragment*". Com isso o mapa será exibido na sua *activity* ou *fragment.*

A classe *GoogleMap* é responsável por manipular um mapa programaticamente, ou seja, atribuir métodos para incluir *markers* (pontos no mapa), mudar posição, mudar zoom, etc. Ela deve receber um objeto do tipo *SupportMapFragment* conseguido com o método *findFragmentById* da classe *SupportFragmentManager* (de modo a suportar versões antigas do Android).

2.3. JSON

JavaScript Object Notation (JSON)[4] é um formato de texto utilizado para a serialização de dados estruturados e é originário dos objetos literais do JavaScript.

JSON pode representar quatro tipos primitivos (*strings*, números, *booleanos* e *null*) e dois tipos estruturados (objetos e *arrays*). Um objeto JSON é uma coleção desordenada de zero ou mais pares nome/valor, em que o nome é uma *string* e um valor pode ser uma *string*, número, *boolean*, *null*, objeto ou *array*. Um *array* JSON é uma sequência ordenada de zero ou mais valores. Tais termos (objeto e *array*) vieram das convenções da linguagem JavaScript.

2.3.1. Gramática do JSON

JSON é uma sequência estruturada de tokens, dos quais temos seis tipos de caracteres estruturais, *strings*, números e três nomes literais. Um JSON pode ser um *JSON Object* ou um *JSON Array.*

Os seis caracteres estruturais são mostrados na Figura 6.

[4] http://www.json.org

Figura 6 - Caracteres estruturais JSON

```
begin-array     = ws %x5B ws  ; [ left square bracket

begin-object    = ws %x7B ws  ; { left curly bracket

end-array       = ws %x5D ws  ; ] right square bracket

end-object      = ws %x7D ws  ; } right curly bracket

name-separator  = ws %x3A ws  ; : colon

value-separator = ws %x2C ws  ; , comma
```

Fonte: Crockford, 2006.

O caractere "[" é utilizado para dar início a um *array* JSON, enquanto o caractere "{" é utilizado para dar início a um objeto JSON. De forma contrária, o caractere "]" é utilizado para encerrar um *array* JSON, ao passo que o caracatere "}" é utilizado para encerrar um objeto JSON. Os dois últimos caracteres, que são ":" e "," são utilizados para separar nomes e separar valores, respectivamente.

JSON se torna amplamente utilizável quando se deseja realizar troca de informações entre componentes diferentes de um sistema, como por exemplo, um servidor (implementado em Python/Django) recebe informações do seu Banco de Dados, e deseja retornar essa mesma informação para dispositivos Android, iOS e Web. Com o JSON a troca de informações se torna prática e homogênea.

2.4. Protocolo REST

Na busca pelas melhores práticas nos estilos de arquiteturas existentes para compor um novo estilo, (FIELDING, 2000), idealizou o protocolo Representational State Transfer, o qual ficou conhecido por REST. Essa técnica de engenharia de software foi desenvolvida para sistemas hipermídia distribuídos como a World Wide Web.

As principais características do REST são:

. **Recursos**: Parâmetros a serem utilizados pelo *Web Service* vindos da requisição. Geralmente esses parâmetros são passados na própria URL;

. **Ações**: O HTTP por si só já provê uma série de métodos que podem ser utilizados, sendo os mais utilizados o POST (geralmente utilizado em login), PUT (cria um recurso), GET (busca um recurso), DELETE (apaga um recurso).

. **Conteúdo**: Alguns protocolos utilizam MIME TYPES para identificar que tipo de conteúdo esta sendo negociado. A identificação geralmente é feita no cabeçalho pelo campo cujo nome é Content-type.

O REST é resultado das melhores práticas dos seguintes estilos:

. **Cliente / Servidor**: O servidor disponibiliza um conjunto de serviços e o cliente faz uso desses serviços;

. **Sistema em Camadas**: Cada camada conhece apenas a interface da camada superior. As camadas intermediárias podem ser utilizadas para melhorar a escalabilidade do sistema, permitindo o balanceamento de carga de serviços, através de múltiplas redes;

. **Cache**: Arquitetura que evita desperdício de banda com a utilização de Proxy HTTP, que armazena as páginas solicitadas pelo cliente;

. **Sem estado de conexão (*Stateless*)**: Nessa arquitetura o servidor não armazena nenhuma informação do contexto. Toda informação necessária para atender a uma requisição deve estar contida nela mesma, tornando o serviço mais simples, não precisando levar em consideração o contexto atual para tomar decisões.

2.5. Lei de Publicidade na Advocacia

De acordo com a OAB (Ordem Brasileira dos Advogados), tendo como base a Lei n. 8906/94, o Código de Ética e Disciplina, o Regulamento Geral e em especial o Provimento 94/2000, a prática de publicidade pelos advogados não é permitida de maneira mercantilista. Isso se dá pelo fato que a OAB considera os serviços prestados pelo advogado, uma atividade não mercantil.

O Provimento nº 94/2000, editado pelo Conselho Federal da Ordem dos Advogados do Brasil, em 05.09.2000, formula de maneira clara o que fica permitido como publicidade para os advogados. O Art 1º do provimento mencionado, refere-se à publicidade informativa, e explica o que seria esta publicidade para o advogado.

Art. 1o. É permitida a publicidade informativa do advogado e da sociedade de advogados, contanto que se limite a levar ao conhecimento do público em geral, ou da clientela, em

particular, dados objetivos e verdadeiros a respeito dos serviços de advocacia que se propõe a prestar, observadas as normas do Código de Ética e Disciplina e as deste Provimento.

O art 2º, por sua vez, define o que é publicidade informativa de maneira enumerativa e taxativa, fazendo assim com que não se tenha possibilidade de ocorrerem outras interpretações.

Art. 2o. Entende-se por publicidade informativa:

a) a identificação pessoal e curricular do advogado ou da sociedade de advogados;

b) o número da inscrição do advogado ou do registro da sociedade;

c) o endereço do escritório principal e das filiais, telefones, fax e endereços;

d) as áreas ou matérias jurídicas de exercício preferencial;

e) o diploma de bacharel em direito, títulos acadêmicos e qualificações profissionais obtidos em estabelecimentos reconhecidos, relativos à profissão de advogado (art. 29, §§ 1o e 2o, do Código de Ética e Disciplina);

f) a indicação das associações culturais e científicas de que faça parte o advogado ou a sociedade de advogados;

g) *os nomes dos advogados integrados ao escritório*

h) o horário de atendimento ao público;

i) os idiomas falados ou escritos.

O art 5º específica os meios possíveis para que este tipo de divulgação da profissão do advogado possa ocorrer sem comprometimento do código de ética.

Art. 5o. São admitidos como veículos de informação publicitária da advocacia

a) Internet, fax, correio eletrônico e outros meios de comunicação semelhantes;

b) revistas, folhetos, jornais, boletins e qualquer outro tipo de imprensa escrita;

c) placa de identificação do escritório;

d) papéis de petições, de recados e de cartas, envelopes e pastas.

Parágrafo único. As páginas mantidas nos meios eletrônicos de comunicação podem fornecer informações a respeito de eventos, de conferências e outras de conteúdo jurídico, úteis à orientação geral, contanto que estas últimas não envolvam casos concretos nem mencionem clientes.

Com base nestes dois artigos do provimento mencionado, fica clara a determinação da limitação existente do uso da publicidade pelo advogado. Sendo assim isso

motiva a criação do aplicativo proposto neste trabalho, porém delimita algumas funções que o aplicativo poderia ter, como por exemplo a opção de avaliar um advogado.

2. TRABALHOS RELACIONADOS

Nesta seção serão apresentados alguns trabalhos relacionados a este, apresentando o foco do trabalho e as suas semelhanças e diferenças em relação a este.

O trabalho proposto por (MARÇAL et al., 2013) busca apresentar uma solução para a evasão do curso de Geologia da UFC. Para tal, foi feita uma parceria com o Departamento de Computação a fim de desenvolver um aplicativo para auxiliar as aulas de campo do curso, e deixá-las mais interativas e dinâmicas. O aplicativo trazia ferramentas de medição de rochas, permitia escrever as características de um ponto específico, além de fazer uso do georreferenciamento na marcação desses pontos e na criação do caminho percorrido durate a aula, para que posteriormente pudessem ser observados no Google Earth.[5] O trabalho de (MARÇAL, 2013) assemelha-se a este pois faz uso do georreferenciamento para melhorar e deixar mais dinâmico um serviço existente, no caso as aulas do curso de Geologia.

(EGGEA, 2013) desenvolveu uma aplicação para dispositivos móveis para servir como um guia turístico virtual para localizar facilmente a posição atual do dispositivo e dos pontos turísticos da cidade de Curitiba. O aplicativo mostra o mapa da cidade com os principais pontos turísticos da cidade de forma simples, clara e objetiva. O turista ao clicar no ponto turístico receberá informações relevantes sobre o local. Também será possível realizar uma busca rápida de pontos turísticos através de uma lista e poderá visualizar o itinerário da linha turismo e seus respectivos pontos de parada. Inicialmente o aplicativo está disponível em português podendo ser estendido a outros idiomas. O trabalho de (EGGEA, 2013) assemelha-se muito a este pois faz uso de georreferenciamento para apresentar alguns pontos para o usuário. Enquanto este trabalho mostra pontos relativos à advogados, e em um nível nacional, o trabalho proposto por (EGGEA, 2013) apresenta pontos turísticos, e relativos apenas a cidade de Curitiba.

O projeto apresentado por (UDDIN *et al*., 2013) busca desenvolver um aplicativo para plataforma Android que utiliza a localização (latitude e longitude) fornecida pelo *smartphone* do usuário e salva essas informações continuamente em um servidor, para que posteriormente, a pessoa responsável pelo gerenciamento dessas informações possa ter controle da posição e caminho percorrido pelos seus usuários, fazendo uso de uma aplicação web de administração dessas informações. Esse sistema se torna interessante quando, por exemplo, um usuário faz parte de um grupo em um *tour*. Com o aplicativo desenvolvido pelo

[5] https://www.google.com/earth/

autor, fica fácil para o responsável pelo grupo, saber exatamente onde cada um dos integrantes está, e assim evitar que eles se percam. Caso algum participante se perca, o administrador tem acesso a última posição deste usuário e assim poderá achá-lo facilmente.

O trabalho apresentado por (UDDIN *et al*., 2013) assemelha-se a este pelo fato de utilizar o GPS do Android como base de uma aplicação real e utilizável. Utiliza o georreferenciamento de cada usuário e exibe informações relativas a esta posição. Diferencia-se pelo fato de as informações de posição do usuário serem utilizadas por outra pessoa, em vez do próprio usuário.

O trabalho proposto por (CHENG *et al*., 2013) busca prover um mecanismo de recomendação de pontos para o usuário baseado não somente na sua localização, mas também nas escolhas feitas por este usuário anteriormente. Assim, as recomendações se tornam mais pessoais e utilizando o mecanismo descrito no trabalho, o processamento se torna mais leve. Com o estudo do trabalho de (CHENG *et al.*, 2013) fica possível saber para onde o usuário quer ir depois, seja depois de jantar em um local específico, seja daqui a dois meses para fazer compras, etc. Com isso, além de tornar as recomendações mais pessoais, acrescenta-se a possibilidade de marketing baseado em localização e costumes do usuário, tornando este estudo relevante para pesquisa e indústria.

Embora o trabalho proposto por (CHENG *et al.*, 2013) forneça pontos baseados em costumes do usuário para saber a intenção de onde ele quer ir, este trabalho assemelha-se pois ambos fazem uso de *Location Based Services* e podem promover o uso de estratégias de marketing.

A seguir, na Tabela 1, é apresentada uma comparação entre os trabalhos apresentados.

Tabela 1 - Comparação entre os trabalhos

	(MARÇAL et al., 2013)	(EGGEA, 2013)	(UDDIN *et al.*, 2013)	(CHENG *et al.*, 2013)	Este trabalho
Desenvolvimento de aplicativo	x	x	x		x
Uso de georreferenciamento	x	x	x	x	x
Criação de um algoritmo				x	
Domínio	Alunos de Geologia	Turistas	Grupo em *tour*	Usuários em geral	Advogados
Sistemas Operacionais Suportados	Android	Android	Android	-	Android, iOS
BackEnd Web	Não possui	Possui	Possui	-	Possui
Nativo	x	x	x	-	x
Acesso a serviço de terceiros					x

Fonte: próprio autor

3. PROCEDIMENTOS METODOLÓGICOS

Nesta seção serão explicados os passos executados para atingir os objetivos deste trabalho:

a) Definir as ações e tipos de busca do aplicativo (requisitos).

   Nesta fase, as ações que um usuário poderia realizar com o aplicativo foram definidas. Tais ações foram pensadas de modo que o usuário pudesse encontrar advogados da melhor maneira possível, ou seja, de maneira fácil e prática.

b) Desenvolver o layout da aplicação.

   Uma aplicação com um design bem pensado é essencial para que se tenha sucesso. Nessa etapa foi definido o padrão de design do Android a ser seguido e foram desenvolvidas as telas da aplicação.

c) Implementar as *activities* e *fragments* da aplicação.

   Nesse momento, foram desenvolvidas todas as *activities* e *fragments* do *app*, responsáveis por pegar os dados do servidor e mostrá-los na tela.

d) Obter a localização do usuário.

   Um dos pontos principais e essenciais desta aplicação é a localização do usuário. Para o Easy Advogados existe a necessidade de possuir o valor de latitude e longitude de um usuário para que o servidor possa retornar os advogados próximos, além de exibir sua posição no mapa.

e) Comunicação com o *Web Service.*

   Após implementar as telas, *activities e fragments* e obter a localização do usuário, é necessário realizar a comunicação com o *Web Service* desenvolvido para que o aplicativo possa obter as informações necessárias.

f) Definir um banco de dados local para trabalhar *off-line.*

   Para que a aplicação se torne mais interessante, é necessário mostrar dados para o usuário mesmo que ele não possua acesso a internet. Para tal, faz-se necessário o uso do SQLite do Android.

g) Avaliação e análise dos resultados.

   Verificar se o trabalho realmente cumpriu os objetivos com base em um questionário realizado com pessoas que testaram aplicativo. Analisar os resultados do questionário para servir de base para possíveis melhorias.

4. EASY ADVOGADOS

O aplicativo Easy Advogados está implementado e funcional. O aplicativo foi desenvolvido para as plataformas Android e iOS. Neste trabalho será abordado o desenvolvimento do aplicativo na plataforma Android, visto que o projeto foi desenvolvido por uma equipe, cada um tendo sua parte no desenvolvimento, em que o autor deste trabalho foi responsável pelo desenvolvimento da versão Android, parte do *back-end* e design da aplicação.

Nesta seção, será explicado detalhadamente como foi realizado o desenvolvimento do aplicativo na plataforma Android, bem como funciona o *Web Service* da aplicação.

4.1. REQUISITOS DA APLICAÇÃO

Os requisitos da aplicação foram definidos através de reuniões presenciais (com os *stakeholders* do projeto) e via *hangout* e presencial(entre os membros da equipe). Os requisitos definidos com os *stakeholders* do projeto foram bem simples e diretos: 1) o aplicativo deveria mostrar em um mapa e em uma lista todos os advogados, previamente cadastrados, que estivessem próximos ao usuário; 2) ao clicar em um advogado no mapa ou na lista de advogados, deveria ser aberta uma nova tela contendo maiores informações sobre o advogado; 3) na tela de detalhes de um advogado, o usuário teria várias informações de um advogado, tais como: nome, foto, endereço, telefone, email, áreas de atuação e uma descrição.

Tais ações foram definidas com os *stakeholders* do projeto. De modo a agregar mais valor para a aplicação, em reunião com a equipe de desenvolvimento, foram definidos mais requisitos para aplicação: 1) deveria primeiramente exibir um mapa e uma lista com os advogados próximos, deveria ter uma opção de busca, sendo ela por área de atuação e por nome do advogado; 2) ao visualizar os detalhes de um advogado, um usuário poderia realizar mais 5 ações além de somente ver as informações do advogado, que seriam: telefonar para o advogado, enviar um email, criar uma rota até o advogado, marcar como favorito este advogado e ver uma galeria de fotos do advogado (ambiente, eventos, etc); 3) como opção presente na tela principal do aplicativo, um usuário poderá também ir para uma tela de visualização de processos, onde poderá digitar o número do seu processo (de primeira instância) e receber o andamento do processo na tela. Este último requisito foi pensado

visando conquistar mais usuários para o aplicativo, fornecendo uma função extra e útil, 4) como função extra para os advogados, o aplicativo realizaria busca de pontos da CAA (Caixa de Assistência dos Advogados), nos quais os advogados tem desconto, mas apenas para advogados cadastrados no sistema, 5) o aplicativo deveria trabalhar com dados *offline*, ou seja, lidar com a falta de internet, desse modo foi decidido que quando o usuário estivesse sem conexão à Internet, ele mostraria dados já salvos no *SQLite* do Android, 6) em uma tela de configurações o usuário poderá escolher se o mapa deve ser exibido e também definir o seu raio de busca (em km).

Por se tratar de uma *startup* os membros da equipe de desenvolvimento também acabam se tornando *stakeholders* do projeto. Assim, nas reuniões da equipe de revisão dos requisitos, a ideia de realizar buscas por pontos da CAA acabou sendo descartada, pois tiraria o foco em um nicho de mercado específico, deixando a ideia essencial do aplicativo misturada com a ideia para outro nicho de mercado.

Como requisitos não funcionais, apenas os membros da equipe definiram tais requisitos, que são: 1) a aplicação deve seguir os padrões de design de cada plataforma (Android e iOS), 2) a aplicação deve ser intuitiva para o usuário e facilitar a obtenção de dados, 3) quando houver necessidade de o usuário esperar, aplicação deverá informar isso com um ícone ou animação, 4) a aplicação deverá manter um bom *layout* tanto em *smartphones* quanto em *tablets.*

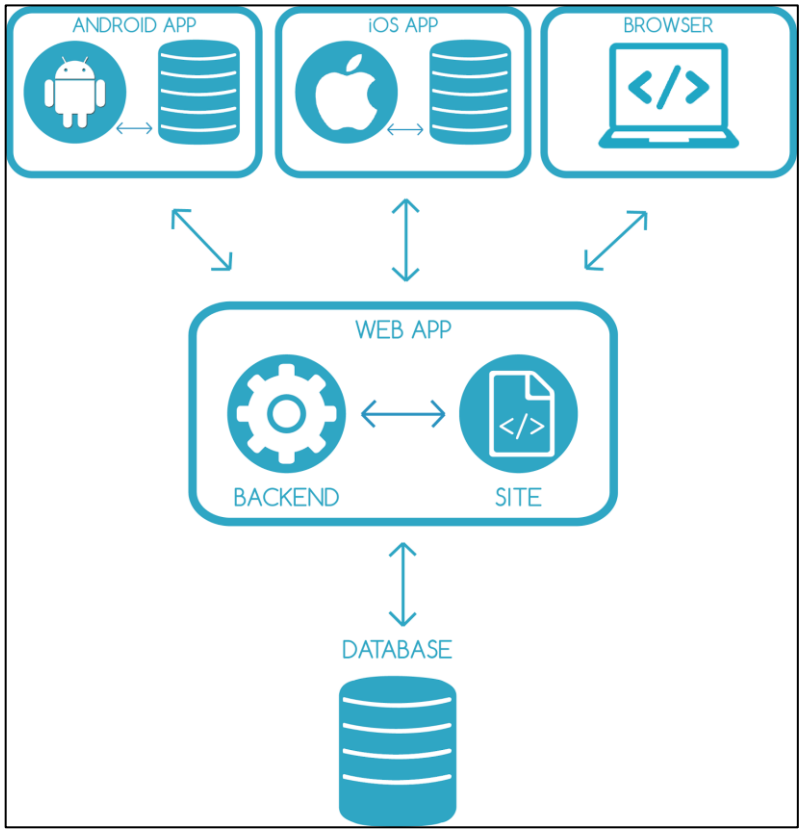
4.1.1. ARQUITETURA GERAL DO SISTEMA

Nas figuras 7 e 8 podemos ver, respectivamente, a arquitetura geral da aplicação e um diagrama de pacotes interno da aplicação Android. Na arquitetura geral podemos como estão estruturados os diferentes componentes da aplicação como um todo, incluindo BackEnd, Banco de Dados e as aplicações Android e iOS. O componente WebApp contém o BackEnd que é responsável por tratar os dados recebidos do Banco de Dados, transformá-los em um JSON e transmitir para as demais aplicações (site, Android e iOS). As aplicações Android e iOS por sua vez, ficam responsáveis por receber estes dados, mostrar para o usuário e salvá-los no banco de dados local para usar futuramente.

No diagrama de pacotes da aplicação Android, podemos ver os diferentes pacotes da aplicação e como eles se relacionam. Nesse diagrama temos como pacote “ui” que contém as classes de visualização, que são *MainActivity, ActivityPrincipal, ActivityAdvogadoDetail, Processo, Contato, Galeria, Galeria Aberta, Configuracoes FragmentNavigationDrawer,*

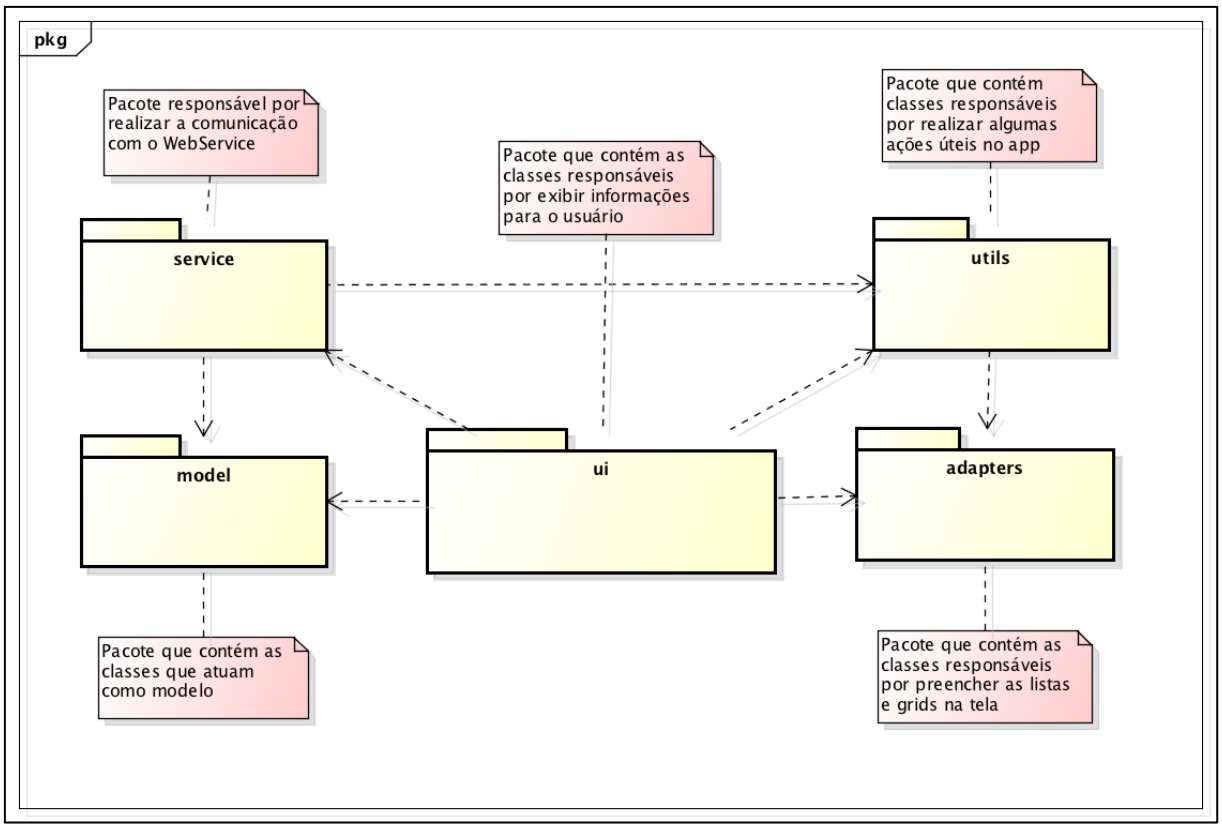
FragmentListaAdvogados e FragmentProcesso. O pacote "*service*" contém as classes de conexão com o servidor que são *AdvogadoWS, ProcessosWS e FotosWS.* O pacote "model" contém as classes de modelo a serem usadas pelo app Android, que são *Advogado, Icon, ItemNavigationDrawe, DataBaseHandler e Fotos.* O pacote "*util*" contém as classes *StackBlur, ServiceGPS e TestConnection*. Por fim o pacote "*adapters"* contém as classes *AdapterGaleria, AdapterGaleriaAberta, AdapterNavigationDrawer, AdapterListaAdvogados e AdapterProcessos.* Na figura podemos ver os relacionamentos, em que o pacote "*ui*" utiliza objetos do pacote "*service*", "*util*", "*model*" e "*adapters*". E o pacote "*service"* utiliza objetos dos pacotes "*util*" e "*model*".

Figura 7 - Arquitetura geral da aplicação



Fonte: próprio autor

Figura 8 - Diagrama de pacotes da aplicação Android



Fonte: próprio autor

4.2. DESENVOLVIMENTO DO LAYOUT

O aplicativo é composto de 8 telas, que são: Splash, Principal, Detalhes, Galeria, Foto Aberta, Contato, Configurações e Ver Processos.

A tela de Splash é chamada logo quando a aplicação é lançada, servindo apenas como uma tela de boas vindas, contendo a logo do aplicativo e é finalizada após três segundos, chamando assim a Principal. A tela Principal contém um menu lateral com opções que o usuário pode escolher, além de contar com um mapa (GoogleMaps) e, abaixo do mapa, uma lista com os advogados próximos. A tela de Detalhes contém detalhes de um advogado escolhido na lista, exibindo a foto e o nome do advogado sobre uma foto de *background*. Logo abaixo é mostrada a área de atuação, endereço, telefone e email. Depois disso são apresentadas duas opções: “Ver Fotos” e “Contato”. Abaixo pode-se ver a localização do advogado no mapa tendo também a opção de criar rota até o escritório do advogado. Na tela

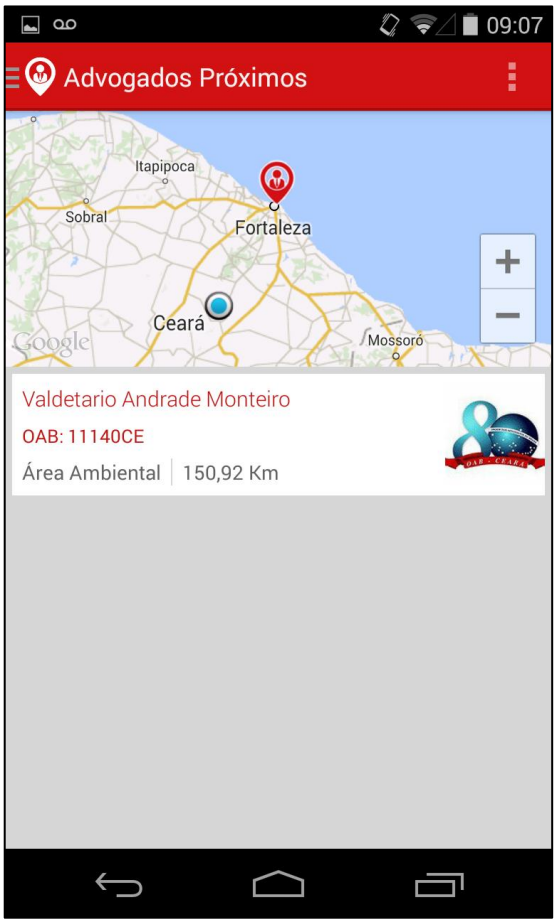
de Galeria, temos as fotos do advogado exibidas em um *GridView* , em que ao clicar em um item, essa foto é expandida. Em Contato, o usuário pode enviar um email ou chamar a tela de ligação, para telefonar para um advogado. Na tela de Processos ele poderá digitar o número do seu processo e receber o andamento deste processo. Por fim na tela de Configurações poderá decidir se o mapa será exibido ou não e o raio de busca por advogados.

A aplicação possui 8 telas que o usuário verá, porém, no desenvolvimento, é necessário desenvolver bem mais do que 8 arquivos de layout, já que precisamos de *layouts* para construir *activities*, *fragments* e até mesmo elementos como *background* de botões, *actionbars* e *layouts* (Linear, Relativo, etc). No Android, esses arquivos de layout ficam dispostos na pasta *res/layout* . Tais arquivos são desenvolvidos utilizando a linguagem XML (*Extensible Markup Language*).

O arquivo *"activity_lista_advogados_inicial.xml"* define o layout principal da aplicação. Este arquivo utiliza o *DrawerLayout* do Android para a contrução da tela, em que este precisa de mais dois layouts para compor sua tela: um layout próprio da tela e o layout do menu lateral, seguindo assim o padrão de estrutura de design mais atual e recomendado do Android segundo (GOOGLE, 2014), o *Navigation Drawer*. No layout da tela Principal, definimos um *Frame Layout* responsável por conter o *fragment "fragment_lista_advogados_inicial.xml",* e esse por sua vez, deve conter um *fragment* para mostrar o mapa do Google, e logo abaixo um *List View* que mostrará os advogados em uma lista.

Alguns arquivos de layouts são usados como um modelo para ser repetido de maneira programática quando se trabalha com *ListViews* (listas) e *GridViews* (grids). Para cada item da lista ou grid, um layout já implementado é utilizado para servir como *layout* base deste item, e exibir suas informações específicas. No Easy Advogados os arquivos precedidos pela palavra "item" são utilizados para este propósito.

Figura 9- Tela principal do aplicativo Easy Advogados



Fonte: próprio autor

Na Figura 9 podemos ver o mapa com o advogado marcado, e abaixo do mapa um *ListView* contendo um item. Este único item foi desenhado utilizando o arquivo *"item_lista_advogados_inicial.xml"* , e este mesmo arquivo será utilizado para os demais itens a serem inseridos nessa lista no decorrer do uso da aplicação.

O arquivo *"activity_settings.xml"* é utilizado quando precisamos criar uma tela de configurações no aplicativo. No caso do Easy Advogados, essa tela dá a possibilidade ao usuário de escolher se quer exibir o mapa ou não, e definir o raio de busca por advogados. O layout desta tela, já é predefinido pelo Android e cabe ao desenvolvedor apenas dizer quais atributos deseja utilizar e como salvá-los. A tela de preferências do Android, salva as definições escolhidas pelo usuário automaticamente utilizando o recurso *SharedPreferences*, que pode ser utilizado em qualquer momento da aplicação.

Figura 10 - Tela de detalhes de um advogado

Fonte: prórpio autor

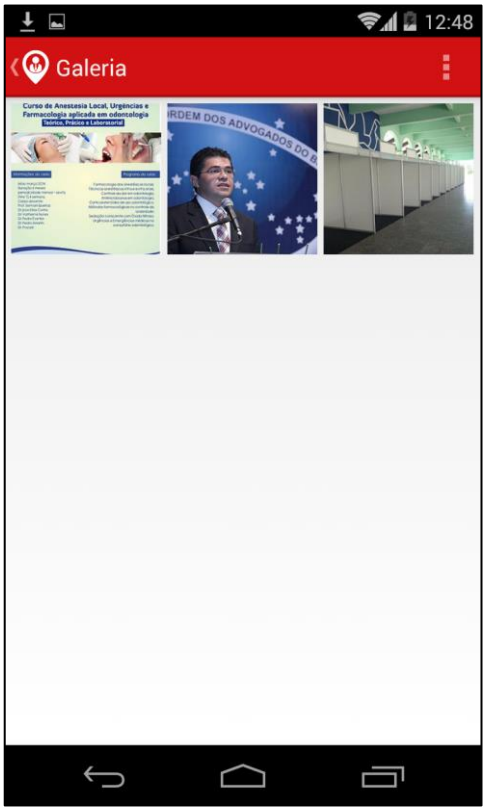
Na tela de Detalhes (Figura 10), o arquivo *"fragment_advogado.xml"* é utilizado para compor o layout dessa *activity*. Nesse layout, é utilizado como base um *Scroll View*, pois devido ao número de informações a serem exibidas, as *views* utrapassam o tamanho da tela, fazendo-se necessário o uso do mesmo. Dentro deste layout temos um *Relative Layout* usado para construir o topo da tela (background, foto, nome e OAB), um *Linear Layout* para construção da parte de informações (área de atuação, endereço, telefone, email e botões de ver fotos e contato), além de outros layouts lineares dentro do primeiro, para que assim possamos ter uma tela mais bem organizada. Logo abaixo temos um *fragment* utilizado para exibir um mapa com a posição do advogado, e um *Text View* sobre o mapa, para receber o evento de click e lançar o aplicativo responsável pelo trajeto, além de termos outro *text view* mostrando uma descrição do advogado.

Figura 11 - Tela de Contato

Fonte: próprio autor

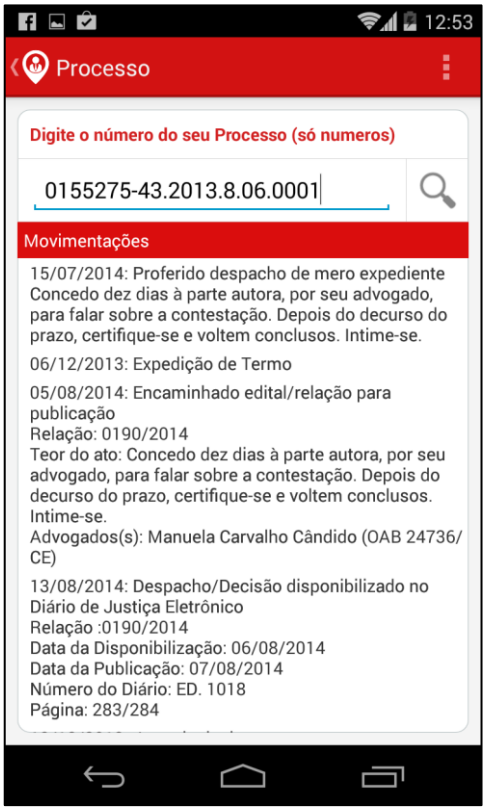
As telas *"activity_contato.xml"*, *"activity_galeria.xml"* e *"activity_processo.xml"* são utilizadas nas telas Contato, Galeria e Processos, respectivamente, como mostra a Figura 11, 12 e 13.

Figura 12 - Tela de Galeria



Fonte: próprio autor

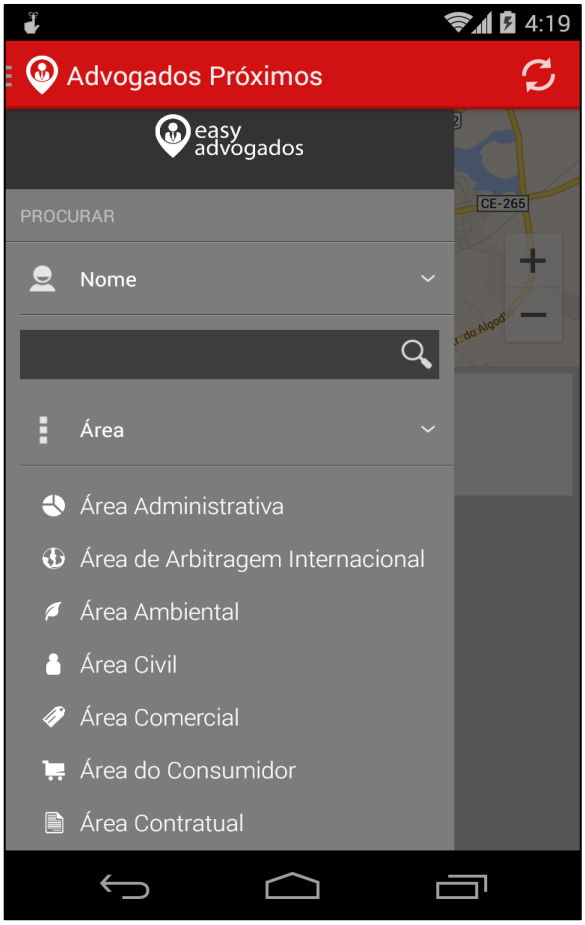
Figura 13 - Tela de Processos



Fonte: próprio autor

Além dos *layouts* utilizados na construção das telas, também são utilizados *layouts (drawables)* para melhorar a composição final da tela, ajudando na criação de *backgrounds* estilizados que são utilizados para dividir melhor a tela, *backgrounds* de botões e *actionbar*. Esses layouts, porém, não se encontram na pasta *"res/layout"*, mas sim na pasta *"res/drawables"*. Essa divisão de pastas é feita para que se tenha um controle melhor do que se é utilizado como *layout* de telas e como *layout* de composição.

Figura 14- Exibição do menu lateral (Navigation Drawer)



Fonte: próprio autor

Um fato importante a ser destacado é o uso do *Navigation Drawer* que, como dito anteriormente, de acordo com (GOOGLE, 2014) é o mais novo e mais recomendado padrão de navegação da plataforma Android. Este padrão possibilita que usemos um *fragment* diretamente como layout do nosso menu, permitindo assim uma maior liberdade na construção podendo adicionar além das opções do menu em si, imagens, campos de texto, etc. Com isso, no Easy Advogados, como podemos ver na figura 14, o menu lateral possui uma

imagem com a logomarca da aplicação, um campo de texto para busca por nome, uma lista com opções relativas a busca por área de atuação, além do acesso as demais funcionalidades e opções.

4.3. OBTENDO A LOCALIZAÇÃO DO USUÁRIO

Como o aplicativo Easy Advogados é um LBS (*Location-Based Service)* faz-se necessário o uso da localização atual do usuário para que o aplicativo funcione da maneira correta. Com a localização do usuário podemos ter os valores de latitude e longitude, que são parâmetros essenciais para a comunicação com o servidor.

Para que possamos obter a localização, precisamos de provedores que possam disponibilizar essa localização do usuário. Esses provedores podem ser o GPS, a *Network* ou ambos. Tais provedores podem estar ativos ou não no *smartphone* do usuário, mas para que primeiramente eles estejam disponíveis para a aplicação, é necessário que no arquivo *AndroidManifest.xml* sejam adicionadas duas permissões, *ACCESS_FINE_LOCATION e ACCESS_COARSE_LOCATION*, que farão com que a aplicação possa utilizar ambos os provedores ou somente um dependendo da disponibilidade.

Para podermos de fato conseguir obter a localização do usuário é necessário implementar uma classe responsável por lidar com essa obtenção de dados. A classe *ServiceGPS.java* é responsável por isso. Esta classe deve implementar a interface do Android *LocationListener*, de modo que ela venha a implementar um método necessário para a obtenção da posição do usuário e outros três métodos que serão usados para fornecer algumas informações no decorrer do uso da classe. Esses métodos são: *onLocationChanged()* (como método principal) e *onStatusChanged(), onProviderEnabled() e onProviderDisabled()* (como métodos de apoio).

Além de implementar os métodos da interface *LocationListener* , viu-se a necessidade de fazer métodos próprios e públicos para serem usados por outras classes que venham a criar instâncias da classe *ServiceGPS.* Métodos como: *ativar()* (que inicia o funcionamento do GPS) e *pausar* (que pára o funcionamento). Esses métodos são necessários pois o serviço de obtenção de localização do Android, assim como também de outros *smartphones*, consomem bastante energia. Desse modo, prover um método público para parar e iniciar um serviço desse tipo, é essencial para preservar a bateria do usuário, fazendo com que seja utilizado somente quando necessário.

Para iniciar a classe, é preciso utilizar um objeto do tipo *LocationManager* que é responsável por requisitar o uso do serviço de localização e pará-lo. No construtor da classe, esse objeto é iniciado e nos métodos de *ativar() e parar()* ele é utilizado, como mostra a figura 15.

Figura 15 - Construtor da classe ServiceGPS e métodos ativar e pausar

```
public ServiceGPS(Context context) {
    this.localizacao = null;
    this.context = context;
    this.lm = (LocationManager) context.getSystemService(Context.LOCATION_SERVICE);
}

public void ativar(){

    Criteria criteria = new Criteria();
    criteria.setAccuracy( Criteria.ACCURACY_FINE );
    criteria.setAltitudeRequired(true);
    String provedor = lm.getBestProvider(criteria, true);

    if ( provedor == null ) {
        Log.d("ServiceGPS", "Nenhum provedor encontrado.");
    } else {
        lm.requestLocationUpdates(provedor, TEMPO_DE_ATUALIZACAO,
                DISTANCIA_DE_ATUALIZACAO , this);
    }

}
```

Fonte: próprio autor

4.4. MODELOS E BANCO DE DADOS LOCAL (SQLITE)

Os modelos utilizados pelo aplicativo ficam localizados no pacote "*br.com.roove.easyadv.model"*. O modelo principal e mais utilizado é o do tipo "Advogado". Alguns *models* estão implementados, porém somente serão usados em trabalhos futuros, como é o caso dos tipos "Escritório", "CategoriaDesconto" e "LojaDesconto". O model "ItemNavigationDrawer" é o utilizado pelo *adapter* responsável por preencher a lista de áreas de atuação no menu lateral da aplicação. O item "Fotos" é utilizado para montar a galeria e possui uma lista de objetos do tipo "Foto". Por fim o item "Processo" é utilizado na amostragem de processos.

A classe Advogado contém os seguinte atributos:

. OAB - Número de inscrição do advogado na Ordem Brasileira dos Advogados;

. Favorito - Indica se o advogado foi marcado como favorito pelo usuário;

. Nome - Nome do advogado;

. Distância - Distância do advogado para o usuário;

. Área de Atuação - Área em que o advogado é especialista;

- Endereço - Endereço onde o advogado está localizado;
- Telefone - Número de telefone utilizado para contato;
- Email - Email utilizado para contato;
- Sobre - Descrição de um advogado;
- UF - Estado no qual o advogado está inscrito;
- Foto do Advogado - Endereço URL da foto do advogado;
- Foto *Background* - Endereço URL da imagem utilizada como *backgroud* da tela de Detalhes;
- Logo - Endereço URL da imagem da logo do advogado;
- Latitude e Longitude - Utilizados para marcar um advogado no mapa e calcular a distância entre o advogado e o usuário;

Para lidar com a falta de conexão com a Internet, um banco de dados local foi implementado. Assim, caso o usuário possua advogados já salvos no seu banco de dados e esteja sem conexão com a Internet, ainda assim os dados serão mostrados para o usuário.

Tais dados *offline* são obtidos quando um usuário possui conexão com a Internet e busca advogados através do aplicativo. Ao encontrar advogados próximos, esses dados são mostrados para o usuário e automaticamente salvos no banco de dados local, para futuramente serem usados em caso de acesso sem conexão.

A classe criada para atender a esta necessidade é a *DataBaseHandler* encontrada no pacote *"br.com.roove.easyadv.model".* Essa classe possui métodos para criar um advogado (*createAdv*), para criar um advogado favorito (*createAdvWithFav*), para trazer todos os advogados salvos localmente (*getAllAdvs*) e para trazer todos os advogados favoritos (*getAllFavs()*).

O método *createAdv()* cria uma advogado tendo como atributo único o seu número de OAB seguido da sigla do estado, assim, quando tentarmos criar um advogado com o mesmo OAB e estado, ao invés de criar um advogado repetido, os dados são atualizados, buscando assim trazer a informação sempre nova e relevante para o usuário.

O banco de dados (caso já não tenha sido criado) é criado logo que a aplicação inicia, bastando apenas que esta instancie um objeto do tipo *DataBaseHandler*, pois toda a parte de criação do banco é feita no seu construtor, e assim o objeto estará disponível para que os métodos mencionados anteriormente sejam usados.

4.5. ACTIVITIES E FRAGMENTS

Com o *layout* da aplicação já implementado e com a classe de obtenção de localização desenvolvida, o próximo passo é a implementação das *Activities* e *Fragments* responsáveis por manipular as *Views* previamente definidas nos arquivos *xml*.

O aplicativo é composto de 9 *activities,* localizadas em "*src/br.com.roove.easyadv.ui",* que são responsáveis por manipular tanto os elementos do layout (*Views*) quanto os *fragments* a serem implementados. As *activities* são: *ActivityAdvogadoDetail.java, ActivityPrincipal.java, Contato.java, Galeria.java, GaleriaAberta.java, MainActivity.java, Processo.java, SettingsActivity.java e UserSettings.java,* sendo que as duas últimas são responsáveis pela tela de Configurações da aplicação, e servem para responder a dispositivos de diferentes versões.

4.5.1. MainActivity

MainActivity.java é a *activity* responsável pela tela de *Splash* (tela inicial) da aplicação. É a primeira a ser chamada e apenas mostra a logo do aplicativo em um *background.* Essa *activity* utiliza o layout "*activity_main.xml"* como seu *layout.* Faz-se uso de um *Handler* para determinar o tempo de visualização da tela de Splash. Este por sua vez passa como parâmetro para o método *postDelayed()* uma nova instância do tipo *Runnable* e um tempo de execução de 3 segundos definido previamente. Após a execução utiliza-se uma *intent* para chamar a próxima tela e a *activity* é finalizada utilizando o método *finish()* do Android, para que da próxima tela não seja possível mais voltar para a tela de Splash.

4.5.2. ActivityPrincipal

ActivityPrincipal.java é a classe chamada após finalizado o *Handler* da anterior. Esta é a *activity* principal de toda a aplicação, nela é manipulado o menu lateral (*Navigation Drawer)* e o *layout* principal da aplicação (*fragment* onde está contido mapa e lista de advogados). O arquivo *.xml* utilizado como *ContentView* dessa *activity* é o arquivo "*activity_lista_advogados_inicial.xml*" que contém tanto o menu lateral quanto as *views* da activity. Aqui fazemos uso da classe *ServiceGPS* criada anteriormente para obtenção da localização do usuário. Primeiramente no método *onCreate(),* todas as *views* utilizadas nessa

activity serão inicializadas e alguns métodos para "escutar" ao clique das *views* serão implementados, como por exemplo os métodos para obtenção da lista de advogados. Tanto os objetos relativos a tela principal quanto ao menu lateral são manipulados nessa *activity*.

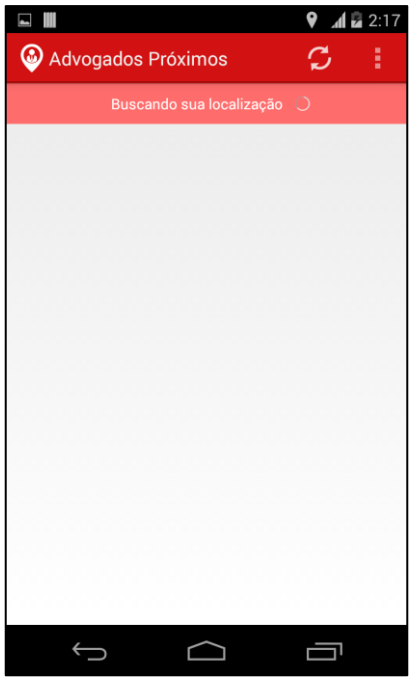
Para que o menu lateral funcione corretamente precisamos de três elementos: uma *view* responsável por conter os elementos desse menu, um *DrawerLayout* responsável lidar tanto com o *layout* do menu quanto da tela principal e um *ActionBarDrawerToggle* responsável por lidar com eventos de clique na *actionbar* (abrir e fechar menu). Com o objeto do tipo *DrawerLayout* podemos abrir o menu, fechar o menu, saber se está aberto ou fechado e etc, mas para isso é preciso passar um objeto do tipo *ActionBarDrawerToggle* como parâmetro para o método *setDrawerListener()*. Sendo assim, precisamos inicializar o nosso objeto desse tipo previamente criado. Na sua inicialização precisamos passar 5 parâmetros para o construtor que são: a *activity* onde ele atuará, o objeto do tipo *DrawerLayout* que o utilizará, um ícone para determinar a informação visual de menu aberto e fechado, e duas *strings* para mostrar textualmente que o menu está aberto ou fechado. Além de passar esses parâmetros, também é necessário implementar dois métodos: *onDraweClosed()* e *onDrawerOpened()*, que são chamados quando o menu é fechado e aberto, respectivamente.

Para aproveitar melhor a tela do *smartphone* e deixar todas as opções do menu já visualmente disponíveis para o usuário, foi necessário utilizar métodos que pudessem esconder e mostrar alguns elementos do menu, como por exemplo a lista de áreas de atuação e o campo de texto de busca por nome. Desse modo, o menu inicialmente é mostrado como na Figura 12. Quando o usuário clica na opção "Nome" e "Área" os elementos aparecem de forma animada, surgindo de cima para baixo. Assim conseguimos mostrar as opções para o usuário de maneira otimizada e com um efeito visualmente interessante. Para isso foram utilizados quatro métodos: *setListViewHighBasedOnChildren(), slideAnimator(), expand() e colapse()*, em que os dois últimos recebem uma *view* como parâmetro e utilizam o terceiro método para animação.

A *ActivityPrincipal* utiliza um *fragment* para mostrar o conteúdo desejado pelo usuário da aplicação. Este componente é chamado logo que a *activity* é criada e preenche todo o *Frame Layout* definido anteriormente no *layout* da *activity*. O *fragment* utilizado pela tela principal é o arquivo *FragmentListaAdvogadosInicial.java* e tem como *layout* o arquivo "*fragment_lista_advogados_inicial.xml*". Esse componente fica responsável por manipular o mapa (adicionando *markers*, modificando zoom, etc) e a lista de advogados (preenchendo a lista com os dados vindos do servidor ou banco de dados local). Também é responsabilidade deste componente sobrescrever o método *onLocationChanged()* da classe *ServiceGPS*, de

modo que se possa saber quando realmente o aplicativo conseguiu pegar a localização do usuário e assim poder exibir informações corretas.

Figura 16 - Obtendo posição do usuário I



Fonte: próprio autor

Figura 17 - Obtendo da posição do usuário II

Fonte: próprio autor

De modo a lidar com a não obtenção da localização do usuário, duas medidas são adotadas para previnir que o aplicativo funcione da maneira errada: 1) O usuário fica impossibilitado de abrir o menu lateral até que sua localização seja encontrada pelo aplicativo, para que assim se evite trabalhar com valores nulos de localização e uma informação é mostrada na tela informando que o aplicativo está buscando a localização do usuário; 2) caso o dispositivo do usuário esteja desabilitado para fornecer a localização (GPS e *Network* desabilitados) um *AlertDialog* é mostrado ao usuário. As duas situações são mostradas nas figuras 16 e 17.

Na *actionbar* dessa tela, foi implementado um novo item, que é utilizado para prover o *refresh* dos dados. Desse modo, o usuário ao clicar no item, chama o método responsável pela comunicação com o servidor.

4.5.3. ActivityAdvogadoDetail

Essa *activity* é chamada logo quando o usuário recebe a lista de advogados e os pontos no mapa e escolhe algum deles, seja clicando em um item da lista ou no *markerInfo* de um ponto do mapa.

Logo que chamada, a *ActivityAdvogadoDetail* recebe os dados da *activity* anterior via *Bundle*. Após receber os dados, preenche esses dados em um objeto do tipo Advogado. Esse objeto será utilizado para fornecer os valores das diferentes *views* presentes na tela, como por exemplo *TextViews* para nome do advogado, OAB, endereço, telefone, email, área de atuação e descrição sobre o advogado. Além das *views* de informações, mais algumas são importantes nessa *activity*, como: *background* e foto do advogado (*ImageView)*, botões relativos a chamada das telas de Galeria e Contato (*Button)*, um mapa para exibir a localização do advogado (*fragment)* e um *TextView* que fica sobre o mapa e ao ser clicado abre um aplicativo relativo a criação de rotas para que seja criada uma rota até o advogado.

Figura 18 - ActionBar da tela de detalhes de um advogado

Fonte: próprio do autor.

Na *actionbar*, um item foi adicionado para que o usuário possa macar como favorito um advogado, como mostra a figura 18, sendo representado por um coração. Ao clicar no ícone, o método de salvar o usuário é chamado. Nesse método utilizamos o objeto do tipo Advogado criado anteriormente para preencher a tela, e o enviamos como parâmetro para o método *createAdvWithFav()* da classe *DataBaseHandler*. Um advogado é considerado favorito pela aplicação quando o atributo "favorito" (int) for de valor diferente de 0. Nesse caso não foi utilizado o tipo booleano, pois o SQlite do Android não trabalha com esse tipo, e assim é aconselhável trabalhar com números inteiros diferentes de zero.

Como *Parent Activity* esta *activity* possui a *ActivityPrincipal* seguindo assim os princípios de design do Android. Assim ao executar os métodos *setHomeButtonEnabled(true)* e *setDisplayHomeAsUpEnabled(true)* poderemos habilitar o ícone *Home*, localizado no lado esquerdo da *actionbar*, e quando o usuário clicar será levado para sua *parent activity*.

4.5.4. Activity Galeria

Após escolher a opção "Ver Fotos" na *activity* anterior, o usuário abre a tela de Galeria, onde poderá ver fotos de um advogado, que o mesmo previamente postou no servidor. Para tal, é necessário implementar três componentes: um *layout* para o item da galeria, um *adapter* responsável por fazer a manipulação desse *layout* preenchendo toda a galeria com as fotos existentes, e um *GridView* que irá conter o *adapter* e os métodos de resposta ao clique.

Essa *activity* utilizará um objeto do tipo *Fotos* enviado pela *activity* anterior via *Bundle*. Para que esse objeto possa ser enviado de uma *activity* para outra, é necessário que ele seja um objeto serializado. Para isso na criação da classe foi necessário implementar a interface *Serializable*. Desse modo, torna-se possível passá-lo como parâmetro através do método *putExtra()* do *Bundle* e recebê-lo na *activity* chamada com o método *getSerializableExtra()*, fazendo um *cast* para a classe Fotos.

4.5.5. Activity GaleriaAberta

Chamada quando um item da Galeria é selecionado pelo usuário, essa *activity* é responsável por mostrar a foto que o usuário deseja ver em tamanho maior. Para isso, foi utilizado um recurso do Android chamado de *ViewPager*. Semelhante ao *GridView*, este também utiliza um *adapter* para montar seus itens e exibi-los para o usuário. No *layout* de um

item foi utilizado um componente, provido por uma biblioteca de terceiros, chamado *"com.ortiz.touch.TouchImageView".* Esse componente provê uma série de recursos para manipulação de imagens quando se deseja trabalhar com uma *ImageView* relativa a uma galeria. Tais recursos incluem *pinch zoom,* toque duplo para *zoom,* salva o estado de *zoom* da imagem de um *pager* para outro, entre outros. Com a ajuda desse recurso, a galeria do EasyAdvogados se torna bastante intuitiva e prática.

4.5.6. Activity Contato

Outra opção a ser escolhida pelo usuário na tela de detalhes de um advogado, é a de "Contato", que por sua vez, chama a *activity* Contato. Esta é responsável por duas funções: 1) fazer uma ligação entre a aplicação e o recurso de envio de email do usuário, possibilitando que ele envie um email para o advogado sem precisar copiar e colar o email do mesmo, 2) facilitar a realização de chamada pelo usuário para um advogado, chamando a tela de *Dialer* do Android diretamente da aplicação e já passando o número do advogado. As duas funções mencionadas acima, são feitas com o uso de *Intents* e recebem da *activity* anterior os dados relativos a email e telefone de um advogado.

4.5.7. Activity Processo

Chamada a partir do menu lateral na tela principal da aplicação, a *activity* de Processos é um recurso interessante para que o usuário tenha uma maior facilidade para acesso ao seus processos de primeira instância do site eSaj[6] (site responsável por armazenar e mostrar andamento de processos).

Essa *activity* faz uso de um *fragment*, que será o recurso responsável por de fato mostrar o conteúdo para o usuário. O uso de *fragments* nessa *activity* foi pensado visando possíveis mudanças na implementação desta tela, pois como existem diversos sites de processos diferentes, pode ser que em trabalhos futuros essa opção venha a ser modificada de modo a atender um número maior de processos. Com isso, o uso destes componentes torna mais fácil e prática a modificação de telas e a forma de uso.

[6] http://esaj.tjce.jus.br/esaj/portal.do?servico=740000

4.5.8. Activity Configurações

A *activity* de Configurações é chamada, também, a partir do menu lateral e é utilizada para prover informações usadas durante toda a aplicação, pois utilizam *SharedPreferences.*

De modo a prover uma sensação de que o usuário está no controle da aplicação, viu-se a necessidade de uma tela de configurações. Essa tela é bem simples de ser utilizada, pois o *layout* é gerado automaticamente pelo Android, cabendo ao desenvolvedor apenas dizer que tipo de recursos a tela deve mostrar, além de onde e como serão salvos e acessados os valores.

No aplicativo Easy Advogados essa *activity* provê a possibilidade de o usuário escolher duas coisas: 1) relativo a Busca, ele pode dizer para a aplicação o valor de busca do raio, variando entre 2, 8, 16 ou 64 Km; e 2) relativo a apresentação do *Layout*, optar por exibir ou não o mapa com advogados na tela principal. Futuramente outras opções poderão ser adicionadas de acordo com a necessidade do aplicativo.

4.6. WEB SERVICE

Para o desenvolvimento do *Web Service* foi utilizado o ambiente de desenvolvimento PyCharm (IntelliJ) e o servidor Web Apache. A linguagem utilizada no desenvolvimento foi Python (2.7). Para criação da aplicação Web foi usado o *framework* Django (1.7). Finalmente, foi utilizada a API Django REST Framework para criar um serviço REST. Essa ferramenta torna o desenvolvimento de Web APIs mais rápido e flexível, incluindo uma API acessível via browser, serialização com suporte para dados ORM (*Object Relational Mapper*) e não-ORM, uma ótima e extensa documentação, além de ser usada e confiada por grandes companhias.

Figura 19 – Exemplo de JSON utilizado pela aplicação


```
[{"id": 2, "name": "Valdetario Andrade Monteiro", "email": "valdetario@gmail.com", "about":
"ALDET\u00c1RIO ANDRADE MONTEIRO \u00e9 natural de Arneiroz, pequena cidade no sert\u00e3o
dos Inhamuns, interior do Cear\u00e1, como bom interiorano \u00e9 afeito a causos e
hist\u00f3rias sobre a imensid\u00e3o do mar azul, dentre outros temas.\r\n\r\nAdvogado
(UNIFOR -1996) e Presidente da Ordem dos Advogados do Brasil, Se\u00e7\u00e3o Caer\u00e1.
\r\n\r\nEx-Secret\u00e1rio Geral da Ordem dos Advogados do Brasil \u2013 Sec\u00e7\u00e3o do
Cear\u00e1, Ex-Presidente da Caixa de Assist\u00eancia dos Advogados do Estado do
Cear\u00e1, Diretor Jur\u00eddico da FENABRAVE/CE, Diretor Jur\u00eddico do SINCODIV/CE e
Professor de Processo Civil da Faculdade de Direito Christus.", "phone_01": "(85)
3268-1561", "phone_02": "", "address": "Rua Visconde de Mau\u00e1, 2230", "uf": "CE",
"position": "-3.7450642,-38.50234230000001", "expertise": [{"id": 1, "name":
"Administrativa", "description": ""}], "oab_code": "11140", "photo_profile": "/static/media
/upload_images/547599_327048907408497_1369721594_n_4.jpg.210x210_q85_crop_detail.jpg",
"photo_cover": "/static/media/upload_images
/165472_402104023236318_345369297_n_5.jpg.210x210_q85_crop_detail.jpg", "photo_logo":
"/static/media/upload_images
/58602_332375556875832_979628124_n_3.jpg.210x210_q85_crop_detail.jpg", "gallery": []},
{"id": 1, "name": "Luis Siqueira", "email": "luissiqueirajr@gmail.com", "about": "",
"phone_01": "8599124916", "phone_02": "8599124916", "address": "", "uf": "CE", "position":
"-4.968507934914592,-39.01652812957758", "expertise": [{"id": 1, "name": "Administrativa",
"description": ""}], "oab_code": "123456", "photo_profile": "/static/media/upload_images
/device-2014-06-23-015823.png.210x210_q85_box-0%2C176%2C480%2C656_crop_detail.jpg",
"photo_cover": null, "photo_logo": null, "gallery": [{"photo": "/static/media/upload_images
/1517587_372347712909297_795569423_n_1.jpg.210x210_q85_crop_detail.jpg", "description":
"Cartaz"}, {"photo": "/static/media/upload_images
/10304358_581932885253430_1835425887312827769_n.jpg.210x210_q85_crop_detail.jpg",
"description": "Valdetario falando"}, {"photo": "/static/media/upload_images
/165472_402104023236318_345369297_n_6.jpg.210x210_q85_crop_detail.jpg", "description": "Foto
01"}]}]
```


Fonte: próprio autor

Para a troca de informações entre cliente e servidor, optou-se por utilizar o formato JSON, que além de ser um protocolo mais simples e compacto do que o XML, ainda permite manipulação de forma nativa pelo Android, a partir do pacote "*org.json".* Decidiu-se retirar do JSON, como mostra a figura 19 alguns espaços que serviriam somente para identação e separação, para que assim se pudesse obter uma resposta mais rápida do servidor.

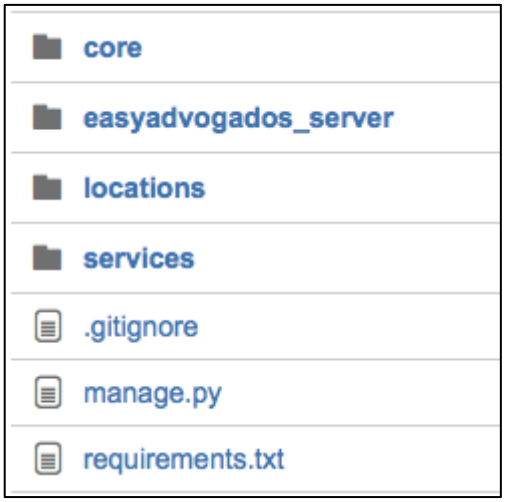
Ao criar um projeto Django, o desenvolvimento de aplicações Web e APIs se torna prático e rápido. Para criação do Banco de Dados do servidor, não é necessário montar diversas consultas e criar diferentes tabelas, pois o próprio *framework* abstrai isso para o desenvolvedor. Desse modo, para criar o banco de dados e os *models* a serem utilizados pelo servidor, um único arquivo necessita ser criado, chamado de *models.py.* Após criar as classes necessárias pela aplicação, o comando "*python2.7 manage.py syncdb"* deve ser executado no servidor para que essas classes sejam transformadas em tabelas no banco de dados. O servidor escolhido para o *deploy* do servidor foi provido pela empresa *WebFaction*[7] e dá suporte a diferentes bancos de dados, mas para este projeto, foi escolhido o *MySQL*.

Uma aplicação Django pode ter várias outras aplicações dentro da mesma, ou seja, isto possibilita que possamos utilizar os mesmos *models* criados anteriormente para responder as requisições de aplicativos (Android, iOS, etc), prover conteúdo para o site do aplicativo e

[7] https://www.webfaction.com

ainda criar de maneira fácil um site administrador para que advogados cadastrados tenham acesso.

Figura 20 - Organização dos projetos da aplicação servidor



Fonte: próprio autor

Os projetos da aplicação *Django Easy Advogados Server* estão organizados como mostra a figura 20. Cada aplicação (pasta) tem uma função específica no *web service*. A aplicação *core* é o núcleo do projeto, ficando responsável pelos *models*, configurações e todos os recursos utilizados de maneira geral pelo sistema. Em *locations* está localizada a aplicação responsável por tratar o LBS do servidor, retornando posições, localizações, etc. A pasta *services* contém as *urls* de serviços a serem utilizadas pelas aplicações (Android, iOS) e site. Por fim, a aplicação *easyadvogados_server* atua como um módulo de configuração do projeto, unindo todas as demais aplicações.

Tabela 2 - URLs de acesso ao servidor

Diretório	**Função**
/api/caace	Retornar pontos de desconto para advogados
/api/profiles	Retornar advogados cadastrados
/api/skills	Retornar área de atuação
/api/skills/id/profiles	Retornar advogados cadastrados por área de atuação

Fonte: próprio autor.

As *urls* do serviço têm diferentes funções e tem como *url* base de toda a aplicação a seguinte: "*easyadvogados.roove.com.br*". As demais formas de acesso a *urls* podem ser vistas na tabela 2, cada uma com sua função característica. A primeira *url* retornar pontos (lojas) em que o advogado tem desconto por ser um advogado da OAB, porém não será usada especificamente neste projeto, somente em trabalhos futuros.

A segunda fica responsável por retornar os advogados presentes no sistema e pode receber alguns parâmetros de modo a ter um retorno diferente, como por exemplo: *"easyadvogados.roove.com.br/api/profiles?name=<nome_do_advogado>"* que retorna os advogados encontrados com o nome digitado pelo usuário e *"easyadvogados.roove.com.br/api/profiles?latitude=<valor>&longitude=<valor>"* que retorna os advogados próximos a um usuário. A terceira retorna todas as áreas de atuação salvas no banco de dados do *Web Service*. A quarta *url* é utilizada para retornar os advogados por área de atuação, em que o id da área de atuação deve ser passado na url, assim como as posições de latitude e longitude também podem ser passadas para restringir a busca por localização (como na *url* 2).

O *Web Service* além de ser responsável por fazer a conversão das informações dos advogados em JSON (e PLIST para iOS) e distribuir o mesmo para as aplicações, tem diversas funções como *trim* do JSON gerado de modo a economizar tempo de requisição, cálculo de distância entre usuário e advogado (retornando a lista por ordem de mais próximo), conversão de informações providas pelo site do eSaj de HTML para JSON e retornar o mesmo para a aplicação, dentre outros mecanismos de flexibilidade por ser hospedado em nuvem.

4.7. COMUNICAÇÃO COM O WEB SERVICE

Com o *Web Service* implementado e funcional, o próximo passo é fazer a conexão entre aplicativo e servidor, para que se possa trabalhar com dados reais. Para isso é necessário trabalhar com requisições HTTP. No caso do EasyAdvogados, apenas requisições do tipo GET são utilizadas com passagem de parâmetros na própria *url.*

Para fazer requisições no Android, precisamos trabalhar de forma assíncrona, ou seja, a aplicação não deve travar e esperar até que a requisição seja completada e o retorno obtido. Para isso, a plataforma provê alguns métodos nativos de requisição HTTP e *AsyncTaks*, que combinados podem prover a necessidade mencionada. Porém, ao trabalhar em um projeto que tem capacidade de crescimento considerável, algumas soluções existentes

podem ser mais interessantes. Pensando nisso, foi decidido fazer uso de uma biblioteca largamente utilizada e recomendada que provê inúmeros métodos e facilidades para requisições HTTP, *download* de arquivos, armazenamento em *cache* e arquivo, manipulação de *views*, além de toda a melhoria desenvolvida ao longo tempo. A biblioteca então utilizada no projeto é a AQuery, que é um projeto *open-source*, podendo ser localizado no repositório do code.google[8].

4.7.1. Activity Principal

As requisições principais acontecem primordialmente no *FragmentListaAdvogadosInicial* que está sendo chamado pela *ActivityPrincipal* como mencionado anteriormente. Este *fragment* fica responsável por dispor os métodos chamados pelo menu lateral, *actionbar* e método *onCreate()* da *activity* principal. Por exemplo, quando a *activity* é primeiramente inicializada, o *fragment* é adicionado na tela e o método da classe *ServiceGPS* é ativado. Ao conseguir a localização do usuário, a aplicação verifica se o usuário possui uma conexão com internet e se possuir o método *getAdvsFromServer()* é chamado, senão, os advogados contidos no banco de dados local são chamados.

Ao chamar o método *getAdvsFromServer*, primeiramente as informações necessárias para serem passadas por parâmetro (raio, latitude e longitude) são atribuídas a variáveis para serem concatenadas na *url* da requisição. Após ter os valores, um objeto do tipo *AQuery* é criado e recebe o contexto da aplicação (*getApplicationContext()*) para que assim, possa receber o método *callBack()* em qualquer momento da aplicação. Para fazer a requisição, utilizamos o método *ajax()* do objeto *AQuery*, passando os seguinte parâmetros: a *url* de acesso, a classe a ser retornada pela requisição (*JSONArray.class* por exemplo) e por fim uma nova instância do *AjaxCallBack<JSONArray>*.

A *url* utilizada pelo método precisa ser montada corretamente, sendo assim, uma medida utilizada para facilitar isso é a criação de uma *static string (URL_GET_ADVS_BASE)* que contém o endereço da *url* básica de acesso. A *string url* é montada da seguinte maneira: URL_GET_ADVS_BASE + "?latitude=" + loc.getLatitude() + "&longitude=" + loc.getLongitude() +"&radius=" + radius + "&format=json".

Na instância do *AjaxCallBack* precisamos implementar o método *callback()* para que possamos manipular os dados retornados (ou não retornados). Esse mérodo traz como parâmetro a *url*, o objeto (*JSONArray*) e um *AjaxStatus*. Com o objeto do tipo *JSONArray*

[8] http://code.google.com/p/android-query

retornado, podemos utilizá-lo para obter as informações referentes aos advogados próximos. Ao recebermos o objeto *JSONArray* utilizamos um laço *for* para percorrer todo o *array* e em cada *loop* criamos um objeto do tipo Advogado que receberá os dados referentes a cada advogado trazido do servidor e ao final deste *loop* o advogado é adicionado ao *ArrayList* que será usado pelo *adapter* para preencher o *ListView*, é utilizado pelo método *addMarkers()* que adicionará um ponto no mapa para cada advogado e é salvo no banco de dados local utilizando o método *createAdv()* da class *DataBaseHandler*. Ao final do *for,* o *arraylist* é passado para o construtor do *adapter* que por sua vez é passado para o *listview* através do método *setAdapter()*. Com isso, os objetos recebidos do servidor são mostrados para o usuário tanto no mapa quanto na lista.

O *layout* de um item da lista principal de advogados é composto de três *TextViews* (nome, OAB , e área e distância) e um *ImageView* (logo). A logo do advogado fica salva em um servidor estático, acessado pelo servidor principal, em que nesse ficam salvas apenas as urls de acesso aquelas imagens. Para que possamos mostrar essa logo para o usuário, é necessário fazer uma requisição ao servidor estático utilizando a *url* fornecida pelo servidor principal. Para isso utilizamos mais uma vez a biblioteca *AQuery*. Ela dispõe de um método, com diversos parâmetros diferentes, para que se possa trabalhar com imagens no Android, possibilitando assim, além de fazer o download da imagem também salvá-la em *cache* e em arquivo, explicitar as dimensões da imagem, preservar o *aspect ratio* da imagem, mostrar uma imagem de previsualização enquanto o download ocorre e ainda fornecer um método de *callback()* para que saibamos quando o download foi finalizado.

Para os outros métodos, *getAdvsFromName()*(que busca advogados baseado num nome) e *getAdvsFromArea()*(que busca advogados por área de atuação), a única mudança fica na *url* a ser passada, visto que a lógica de *download* e *parser* é a mesma.

4.7.2. ActivityAdvogadoDetail

Por se tratar de uma *activity* que já recebe dados da anterior, as requisições feitas ao servidor, são somente para mostrar imagens. Essas imagens são referentes a foto do advogado e ao *background.* No caso da imagem da foto do advogado, ao passarmos os parâmetros para o método do objeto *AQuery*, dois desses parâmetros foram relativos a dimensão da imagem, sendo passados <valor>, para que fosse retornada uma imagem quadrada. Após a imagem ser recebida ela é setada diretamente para o *ImageView* correspondente (passado por parâmetro para o método da biblioteca) e então é exibida.

4.7.3. Galeria

A *activity* responsável por mostrar a galeria de um advogado, assim como a *ActivityAdvogadoDetail*, também só faz requisições ao servidor para download de imagens. Na *activity* Principal um objeto do tipo Fotos é criado para que possa receber a lista de endereços de imagens da galeria vindas do servidor, após preenchido, esse objeto é passado para a *activity* de detalhes de um advogado, que por sua vez passa para a *activity* Galeria. Com a lista de endereços ja em uso na *activity*, um *adapter* é utilizado para preencher o *GridView* com as imagens. A requisição é feita da mesma forma que na *activity* anterior, tendo como diferença nessa, apenas uma imagem que é utilizada em previsualização.

4.7.4. Processo

Para que um usuário possa verificar o andamento do seu processo sem o aplicativo, ele tem que abrir um navegador, acessar o site do eSaj, procurar a aba de processos, digitar o número do processo e só então ele receberá os dados. Com o aplicativo Easy Advogados, basta que a partir do menu lateral, ele abra a *activity* de visualização de processos, digite o número e toque no botão de pesquisa para que assim receba facilmente os dados do processo e as movimentações. Mas para que isso possa acontecer, é necessário que o servidor faça todo trabalho de *parse* das informações e no fim, retorne um JSON com essas informações.

Ao tocar no botão de pesquisa, o aplicativo acessa o servidor, através do método *ajax()* de um objeto *AQuery* passando como parâmetro a *url* base do servidor concatenada com o número do processo a ser buscado e formato JSON (além do objeto de retorno e instância *AjaxCallBack* já mencionados anteriormente). Ao receber a requisição do aplicativo, o servidor acessa o site do eSaj, passando o número recebido pela aplicação na *url*, e recebe uma página em HTML com os resultados da pesquisa. Visto que passar uma página HTML para o usuário não seria eficiente, intuitivo e bonito, a estratégia pensada para contornar esse problema foi a implementação de um *parser* de HTML, que reconhece os resultados retornados pelo site. Como o sistema pode retornar diferentes campos (dados do processo, data, movimentações, etc) e esses dados variam, podendo aparecer ou não, o *parser* foi implementado de modo a reconhecer os diferentes dados retornados, separando-os por categoria, chave e valor. Sendo assim, o JSON retornado poderá variar de acordo com a variação dos valores fornecidos pelo site de processos.

Como a aplicação também não tem controle dos dados retornados pelo servidor, a tela de exibição dos processos para o usuário deveria ser totalmente parametrizada para que assim, independente do JSON recebido, o aplicativo possa exibir as informações de maneira correta e separadas por categoria. Para isso, um *layout* de *TextView* foi criado para preencher os valores de categoria e outro para os valores de "chave-valor" retornados pelo JSON. Para que possamos utilizar o retorno do servidor, devemos mais uma vez, implementar o método *callback()* da instância do tipo *AjaxCallBack* passada por parâmetro para o objeto *AQuery*. A implementação do método, como mostra a figura 21, é feita utilizando um objeto do tipo *Iterator* com as chaves do JSON retornado pelo servidor. Para cada chave que o JSON possui, um *TextView* utilizado para mostrar a categoria é adicionado, tendo como texto a própria chave. Após isso, dentro do primeiro *while*, mais um *while* é utilizado para passar por todas as chaves e valores de uma categoria, e assim, da mesma forma que no primeiro, um *TextView* é adicionado, tendo como valor, a própria chave, seguida de dois pontos, seguida do valor. Com isso podemos garantir que a aplicação funcionará, mostrando o processo, independente do JSON recebido.

Figura 21 - Método de parse do JSON relativo ao andamento de processos

```
@Override
public void callback(String url, JSONObject json, AjaxStatus status) {
    progress.setVisibility(View.GONE);
    try {
        Iterator<String> it = json.keys();
        while (it.hasNext()){

            String key = it.next();
            TextView tv = (TextView)LayoutInflater.from(getActivity()).inflate(R.layout.item_scroll_processo, null);
            tv.setText(key);
            scroll.addView(tv);
            JSONObject obj = json.getJSONObject(key);
            Iterator<String> contents = obj.keys();
            while (contents.hasNext()){
                String key_process = contents.next();
                TextView text = (TextView)LayoutInflater.from(getActivity()).inflate(R.layout.item_scroll_text, null);
                text.setText(key_process + ": " + obj.getString(key_process));
                scroll.addView(text);
            }}
    } catch (JSONException e) {
        // TODO Auto-generated catch block
        Toast.makeText(getActivity(), "Ocorreu um problema. \nTente novamente!", Toast.LENGTH_SHORT).show();
        e.printStackTrace();
    }
}
```

Fonte: próprio autor.

4.8. Conclusão da Seção

Nesta seção foi apresentado o desenvolvimento dos procedimentos metodológicos deste trabalho, explanando o que foi feito, como foi feito, o que foi usado e porque foi feito e utilizado. Esta seção deixa claro como o aplicativo foi desenvolvido e quais técnicas foram

utilizadas**.** Foi falado sobre o layout do aplicativo, banco de dados, *webservice, activities e fragments* da aplicação, bem como suas conexões.

Na seção seguinte será mostrada uma avaliação deste trabalho e se os objetivos do mesmo foram atingidos.

5. AVALIAÇÃO

Esta seção tem como objetivo avaliar o aplicativo desenvolvido. Para isso foi realizado um questionário com pessoas que utilizaram o aplicativo e deram suas opiniões. As perguntas escolhidas tiveram como base os aspectos mais importantes de um aplicativo, tais como design, usabilidade, utilidade e as próprias funções que o aplicativo deve exercer.

5.1. Análise das respostas do formulário de pesquisa

O aplicativo proposto nesse trabalho visa resolver duas questões principais: 1) facilitar a busca por advogados pelos usuários do aplicativo e 2) possibilitar aos advogados um meio de se auto divulgarem sem infringir o código de ética da OAB.

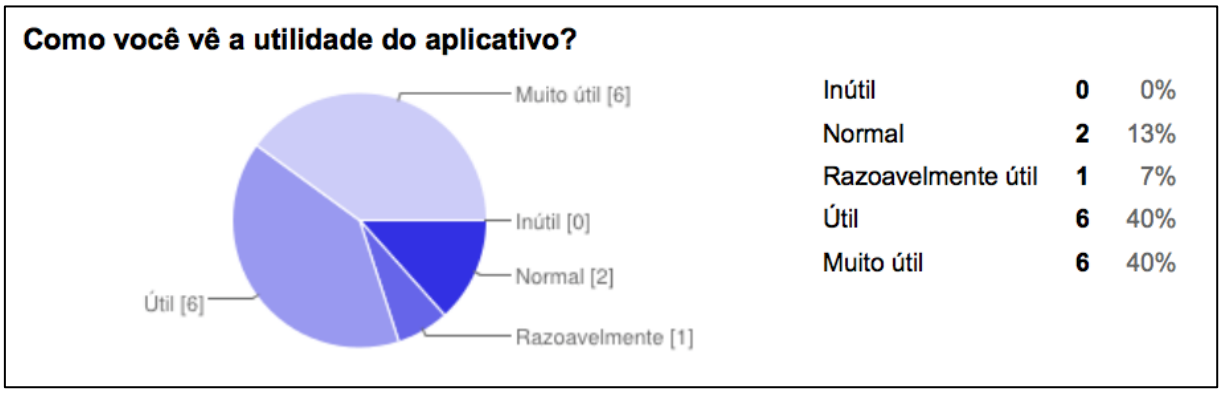
Após a implementação do aplicativo e testes realizados no mesmo, pôde-se comprovar que os resultados foram obtidos, pois o usuário recebia, facilmente em seu smartphone, advogados próximos a ele, e esses advogados conseguiam mostrar suas informações para possíveis clientes de maneira fácil, prática e inovadora.

De modo a proporcionar uma facilidade maior para o usuário no momento de olhar movimentação de processos, o aplicativo deveria também permitir que esses processos fossem visualizados no próprio Easy Advogados. Sendo assim, com o desenvolvimento concluído, os processos foram visualizados no aplicativo de maneira prática e totalmente parametrizada.

Para que a o aplicativo pudesse ser validado de uma maneira mais geral e formal, foi realizado um questionário com 15 pessoas que usaram o aplicativo e analisaram aspectos relativos a interface, usabilidade, tempo de resposta, utilidade, etc. As perguntas relativas ao questionário encontram-se no Apêndice A deste trabalho.

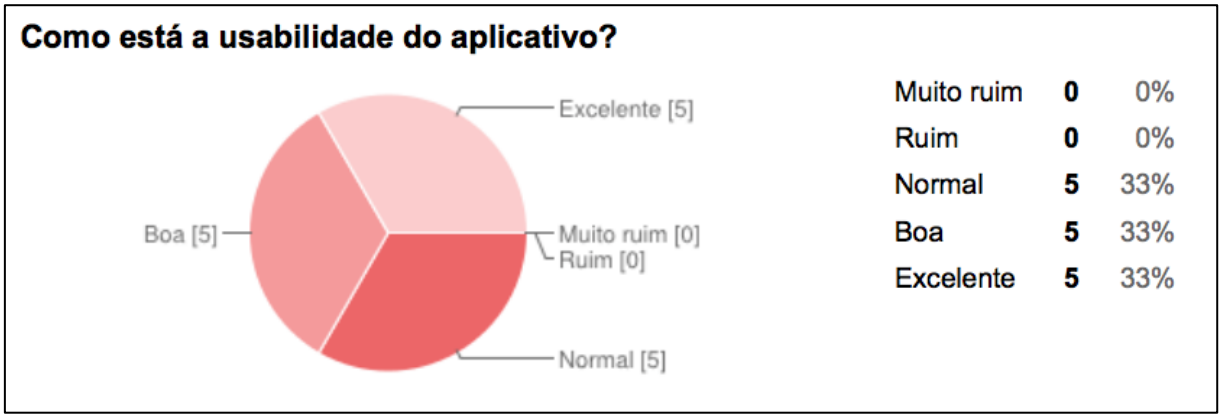
Na figura 22 podemos ver o gráfico de respostas relativo ao aspecto de utilidade do aplicativo. Com base no gráfico podemos observar que a maioria achou o aplicativo útil ou muito útil. Na figura 23 é mostrada a análise das respostas relativas a usabilidade do aplicativo, em que podemos notar que o aplicativo na sua maioria está com a usabilidade boa ou muito boa. Na figura 24 e 25, podemos ver os gráficos relativos às perguntas de busca por nome e área, respectivamente. Nestes gráficos também foi obtida uma análise positiva do aplicativo. Por fim, na figura 26, vemos um gráfico relativo à visualização de processos, que nos mostra que a visualização de processos está boa. Outros gráficos foram gerados para as demais perguntas, os quais também retornaram aspectos positivos do aplicativo.

Figura 22 - Gráfico relativo à utilidade do aplicativo



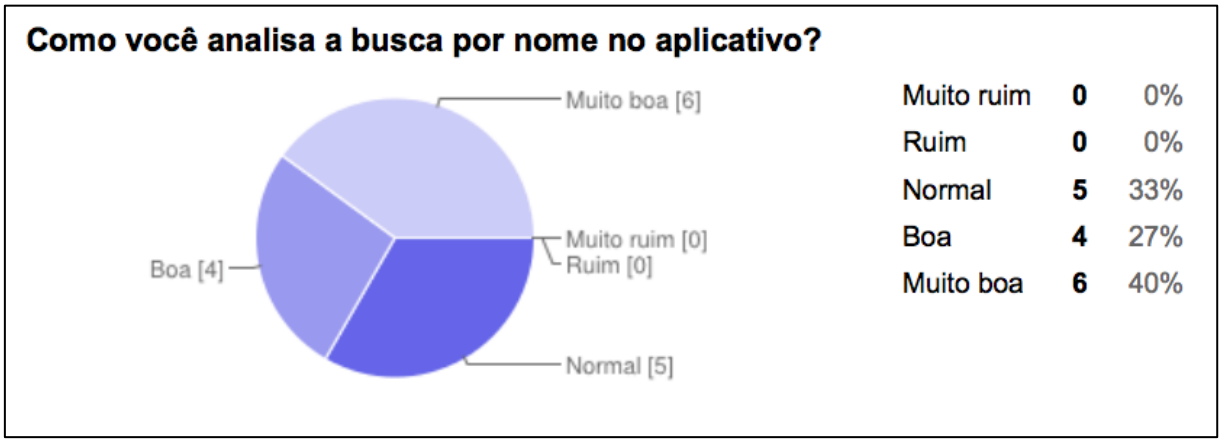
Fonte: próprio autor

Figura 23 - Gráfico relativo à usabilidade do aplicativo



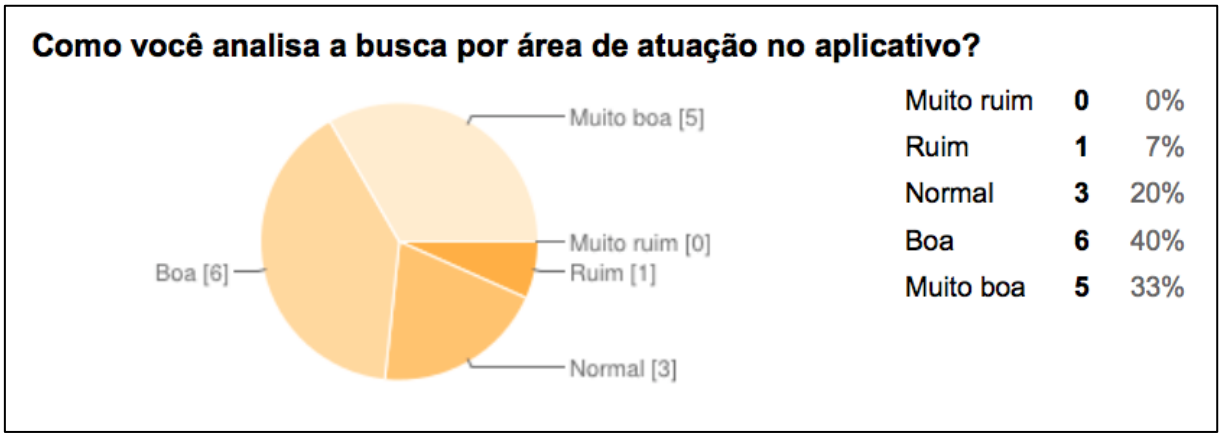
Fonte: próprio autor

Figura 24 - Gráfico relativo à busca por nome do aplicativo



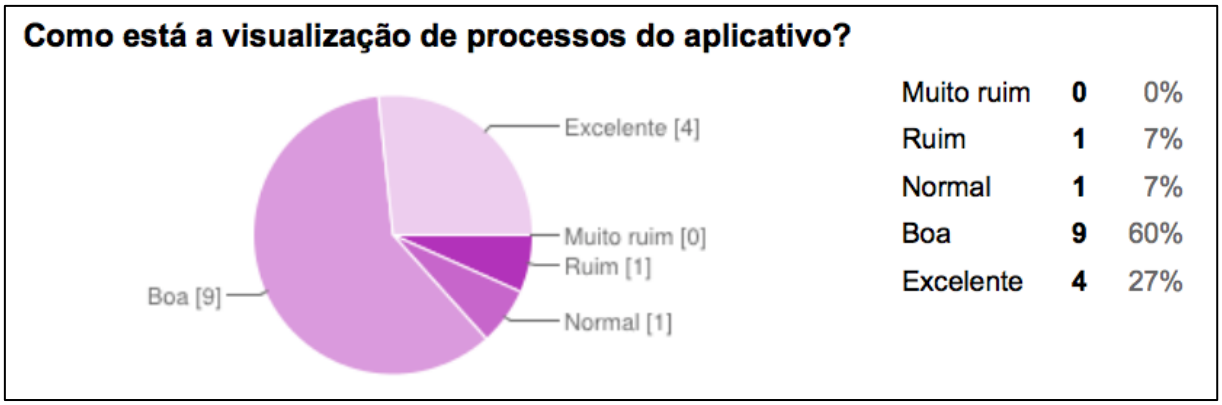
Fonte: próprio autor

Figura 25 - Gráfico relativo à busca por área do aplicativo



Fonte: próprio autor

Figura 26 - Gráfico relativo à visualização de processos do aplicativo



Fonte: próprio autor

Com base nos testes e na pesquisa realizada, pode-se observar que todos os objetivos do trabalho foram concluídos com sucesso, tornando-o assim uma ferramenta útil tanto para quem procura advogados, quanto para advogados que procuram clientes, visto que a grande maioria dos usuários achou o aplicativo útil. Porém, mesmo com os resultados positivos no aspecto de busca de advogados, a ferramenta pode ter melhorias na parte da interação entre usuários e advogados, fazendo com que o aplicativo passe a ter um maior valor agregado.

6. CONSIDERAÇÕES FINAIS

Com o crescente uso de *smartphones* pela população em geral, a quantidade de modelos de negócio na área *mobile* tem crescido cada vez mais. Muitas vezes é necessário saber reconhecer oportunidades e problemas existentes de modo a criar uma solução.

Desse modo, o aplicativo Easy Advogados surge como uma solução para usuários que necessitam de um advogado, para uma determinada área, mas não conhecem e para advogados que desejam se posicionar melhor, em um mercado de trabalho tão competitivo, conseguindo novos clientes de uma maneira inovadora e interativa. Sendo assim, o trabalho proposto buscou desenvolver esta ferramenta que, fazendo uso da tecnologia de georreferenciamento, fornece uma maneira prática para os usuários encontrarem advogados, de uma determinada área, que estão por perto. Além disso, o aplicativo também permite a estes usuários visualizarem o andamento de processos na tela do *smartphone* sem precisar acessar o site responsável.

Como limitações, este trabalho apresenta algumas em relação ao próprio nicho de atuação, pois na advocacia muitas leis acabam por restringir algumas formas de interação que poderiam deixar o aplicativo mais amigável para o usuário e que possibilitariam uma expansão mais rápida, como por exemplo uma forma mais concreta de interação com advogados.

Como trabalhos futuros deste projeto, estão a melhoria na interação do usuário com o software, posteriormente permitindo que o mesmo possa avaliar e fazer comentários sobre um advogado. Também será implementado uma área de fórum, onde usuários poderão fazer perguntas aos advogados, tirando assim, algumas dúvidas que muitas vezes surgem e que os mesmos não sabem onde encontrar a resposta correta.

Com isso, o Easy Advogados passará a alcançar um público cada vez maior, pois aumentará a interatividade entre advogados e futuros clientes, possibilitando assim uma maior aproximação entre os dois lados.

REFERÊNCIAS

BRASIL, Estatuto da OAB – **Lei no 8.906, de 04 de julho de 1994**. Ordem dos Advogados do Brasil – Florianópolis: OAB/SC Editora, 2004.

CHENG, Chen et al. **Where you like to go next: Successive point-of-interest recommendation.** In: Proceedings of the Twenty-Third international joint conference on Artificial Intelligence. AAAI Press, 2013. p. 2605-2611. Disponível em: <http://ijcai.org/papers13/Papers/IJCAI13-384.pdf>. Acesso em: 10 out. 2014.

Douglas Crockford. **The application/json media type for javascript object notation (JSON)**, 2006. Disponível em <http://tools.ietf.org/html/rfc4627.>

EGGEA, Rodrigo Fagundes. Eggea, Rodrigo Fagundes. **"Aplicação Android utilizando sistema de localização geográfica para determinação de pontos turísticos na cidade de Curitiba."** (2014). 2013. 57 f. Monografia (Especialização) - Curso de EspecializaÇÃo em Tecnologia Java e Desenvolvimento Para Dispositivos MÓveis, Departamento de Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná., Universidade Tecnológica Federal do ParanÁ, Curitiba, 2013. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/2484/1/CT_TECJAVMOV_I_2012_10.pdf>. Acesso em: 10 out. 2014.

FERRARO, Richard; AKTIHANONOG, Murat. **Location Based Services Past, Present and Future.** Green paper from Location Based Services Navigating through the Mobile Jungle, janeiro, 2010.

FIELDING, R. **Architectural Styles and the Design of Network-based Software Architectures.** Tese (Doutorado) — University of California, Irvine, 2000.

GOOGLE: **Android Developers**. Disponível em: <http://developer.android.com> Acesso em 10 de outubro

KUPPER, Axel. **Location-based Services: Fundamentals and Operation.** John Wiley & Sons Lt

LECHETA, Ricardo R. **Google Android-3ª Edição: Aprenda a criar aplicações para dispositivos móveis com o Android SDK. 3**. ed. São Paulo: Novatec Editora, 2013. 31 p

MARÇAL, Edgar, et al. **"Geomóvel: Um Aplicativo para Auxílio a Aulas de Campo de Geologia."** Anais do Simpósio Brasileiro de Informática na Educação. Vol. 24. No. 1. 2013.

UDDIN, Md. Palash et al. **GPS-based Location Tracking System via Android Device**. 2013. Disponível em: <http://ijrcee.org/index.php/ijrcee/article/view/101/141>. Acesso em: 10 out. 2014.

APÊNDICES

APÊNDICE A – Perguntas do questionário de avaliação do aplicativo

Como você vê a utilidade do aplicativo?
Você acha que o aplicativo poderá ser útil para a população?

- Inútil
- Normal
- Razoavelmente útil
- Útil
- Muito útil

Como está o tempo de resposta do aplicativo?
O aplicativo responde rápido a sua requisição por advogados ou processos?

- Muito devagar
- Devagar
- Normal
- Rápido
- Muito rápido

Como está a usabilidade do aplicativo?
Você acha fácil utilizar o aplicativo e intuitivamente sabe onde achar o que busca?

- Muito ruim
- Ruim
- Normal
- Boa
- Excelente

Como você analisa o funcionamento offline do aplicativo?
O aplicativo funciona sem internet da maneira proposta?

- Muito ruim
- Ruim
- Normal
- Bom
- Muito bom

Como você analisa a interface gráfica do aplicativo?
O layout está bem feito? Está bonito? Agrada visualmente?

- Muito feio
- Feio
- Normal
- Bonito
- Muito bonito

Como está a visualização de processos do aplicativo?
Os processos estão aparecendo? Os dados do processo são mostrados?

- Muito ruim
- Ruim
- Normal
- Boa
- Excelente

Como você analisa a busca por nome no aplicativo?

Buscar um advogado por nome é fácil? É útil? É rápido?

- Muito ruim
- Ruim
- Normal
- Boa
- Muito boa

Como você analisa a busca por área de atuação no aplicativo?

Buscar advogado por área de atuação é fácil, útil, prático e rápido?

- Muito ruim
- Ruim
- Normal
- Boa
- Muito boa

As informações sobre um advogado são suficientes e úteis?

- Pouca informação
- Muita informação desnecessária
- Muita informação útil
- As informações são suficientes

Como você avalia a construção de um fórum no aplicativo, onde usuários poderão postar dúvidas?

Usuários poderão postar dúvidas sobre seus direitos, terem suas dúvidas respondidas por advogados, e assim escolherem um advogado para auxiliar no caso.

- Muito ruim
- Ruim
- Normal
- Boa
- Excelente