Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

Party: Uma aplicação móvel para divulgação de eventos

Gabriel de Biasi

Prof. Dr. Rubens Barbosa Filho (Orientador)

Dourados - MS
2015

Party: Uma aplicação móvel para divulgação de eventos

Gabriel de Biasi

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Gabriel de Biasi e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Dourados, 24 de novembro de 2015

Prof. Dr. Rubens Barbosa Filho (Orientador)

Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

Party: Uma aplicação móvel para divulgação de eventos

Gabriel de Biasi

Novembro de 2015

Banca Examinadora:

Prof. Dr. Rubens Barbosa Filho (Orientador)
Área de Computação – UEMS

Prof.ª Dr.ª Glaucia Gabriel Sass
Área de Computação – UEMS

Prof.ª MSc. Jéssica Bassani de Oliveira
Área de Computação – UEMS

"Ideias ousadas são como as peças de xadrez que se movem para a frente;
podem ser comidas, mas podem começar um jogo vitorioso."
(Johann Goethe)

AGRADECIMENTOS

Primeiramente aos meus pais, Fernando e Egnalda, pelo incentivo e apoio durante todo o período da minha graduação.

Ao meu orientador, Prof. Dr. Rubens Barbosa Filho, pela paciência e dedicação ao dar todas as orientações necessárias para a conclusão deste trabalho.

À todos os professores do curso de Ciência da Computação da UEMS, que durante toda minha graduação tiveram a total dedicação de lecionar todas as disciplinas que me tornaram capaz de criar este trabalho.

E a todos os meus amigos e colegas da universidade que contribuíram diretamente ou indiretamente à este trabalho.

Meu muito obrigado.

Gabriel de Biasi

RESUMO

O objetivo deste trabalho é criar uma aplicação móvel para o sistema operacional Android que permita a busca por eventos que estejam geograficamente próximos ao usuário. Os métodos tradicionais de divulgações como o rádio, televisão, jornais e revistas são considerados muito efetivos, apesar de alguns tipos de produtos hoje em dia não utilizarem mais estes métodos. Algumas empresas que promovem eventos estão investindo em divulgação pelas redes sociais mais famosas, em busca de um alcance maior de público e apostando em um processo de disseminação da informação mais rápido em comparação aos métodos tradicionais. Analisando estes métodos de divulgação tradicionais e buscando evitar a divulgação dos eventos por meio das redes sociais, a ideia de criar um aplicativo móvel que informe exatamente o evento que o usuário busca, usando apenas informações sobre sua localização, seu estilo favorito e sua cidade atual parece ser uma proposta bem interessante. A proposta é que utilizando o aparelho GPS do dispositivo e um banco de dados com extensão geográfica, as buscas por eventos seja feita de maneira automatizada, fazendo uso do middleware GeoDjango que faz a consulta por alcance. Com a implementação de um site de gerenciamento em Django, os promotores de eventos poderão criar eventos para que possam ser divulgados na aplicação. Para verificar a corretude da aplicação e do site de gerenciamento, foram aplicados três estudos de caso onde testou-se a inserção de um novo evento no primeiro caso, a busca de eventos por GPS no segundo caso e a busca por cidade no terceiro caso. Entre os objetivos concluídos deste trabalho estão o estudo do sistema operacional Android, que a partir dos referenciais teóricos foi possível desenvolver a aplicação totalmente funcional, o estudo da linguagem Python que proporcionou o desenvolvimento do servidor de dados do trabalho, a criação da documentação usando os conceitos da UML e a execução completa dos estudos de caso.

Palavras-chave: Android. Java. Aplicativo.

ABSTRACT

The objective of this work is to create a mobile application for the Android system that allows the search for events that are geographically closer to the user. Traditional methods of disclosures such as radio, television, newspapers and magazines are considered very effective, although some types of products nowadays no longer use these methods. Some companies that promote events are investing in disclosure by the most famous social networks, looking for a greater range of public and betting on a dissemination process the information faster compared to traditional methods. Analyzing these traditional dissemination methods and seeking to prevent the disclosure of the events through social networks, the idea of creating a mobile application that accurately report the event the user to search using only information about your location, your favorite style and your current city It seems like a very interesting proposal. The proposal is that using the device's GPS system and a database with geographic spread, searches for events is made in an automated manner, making use of GeoDjango middleware that makes the query by range. With the implementation of a Django site management, event promoters can create events so they can be disclosed in the application. To check the correctness of the implementation and management site, were applied three case studies which tested the insertion of a new event in the first case, the search for events by GPS in the second case and the search for the city in the third case. Among the objectives completed this work are the study of Android operating system, which from the theoretical references it was possible to develop a fully functional application, the study of the Python language that enabled the development of the job data server, creating documentation using the concepts of UML and the full implementation of the case studies.

Keywords: Android. Java. Application.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
GPS	Global Positioning System
GUI	Graphical User Interface
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
MVC	Model-View-Controller
RAM	Random-Access Memory
RF	Requisito Funcional
RNF	Requisito Não Funcional
RU	Requisito de Usuário
SDK	Software Development Kit
UML	Unified Model Language
URL	Uniform Resource Locator
XML	eXtensible Markup Language

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

1 INTRODUÇÃO

Este trabalho propõe o desenvolvimento de um aplicativo móvel para a plataforma Android, onde será possível encontrar informações detalhadas sobre os eventos que estejam em locais próximos aos usuários que tenham o aplicativo em seus dispositivos móveis.

Os métodos tradicionais de divulgações como o rádio, televisão, jornais e revistas são considerados muito efetivos, apesar de alguns tipos de produtos hoje em dia não utilizarem mais estes métodos. Algumas empresas que promovem eventos estão investindo em divulgação pelas redes sociais mais famosas, em busca de um alcance maior de público e apostando em um processo de disseminação da informação mais rápido em comparação aos métodos tradicionais.

Entretanto, apesar das redes sociais conterem uma grande quantidade de usuários, elas não dão garantias que as publicações colocadas nela irão atingir todos os usuários pertencentes da rede ou do grupo em que a publicação foi colocada, fazendo com que o alcance de público não seja tão grande como se esperava.

Analisando estes métodos de divulgação tradicionais e buscando evitar a divulgação dos eventos por meio das redes sociais, a ideia de criar um aplicativo móvel que informe exatamente o evento que o usuário busca, usando apenas informações sobre sua localização, seu estilo favorito e sua cidade atual parece ser uma proposta bem interessante.

1.1 Objetivos

O objetivo principal deste trabalho é desenvolver uma aplicação móvel que permita divulgar eventos a grupos de usuários cujos interesses estejam em afinidade com os eventos divulgados e que estão em locais próximos ao usuários do aplicativo.

Os objetivos específicos deste trabalho são:

a) Estudar o sistema operacional Android;

b) Estudar a linguagem Python para a utilização do framework de desenvolvimento Django;

c) Desenvolver o aplicativo para a divulgação dos eventos;

d) Implementar um painel de controle de eventos;

e) Criar a documentação inicial do sistema.

f) Criar estudos de caso e documentar os resultados.

1.2 Justificativa

Este trabalho justifica-se pela falta de uma aplicação móvel que tenha a mesma capacidade dos métodos divulgação de eventos atuais, observando-se a mudança de perfil das pessoas que vão à eventos com frequência. Uma aplicação móvel seria uma nova solução em comparação aos métodos já citados, porque possuiria a capacidade de aumentar o número de pessoas alcançadas pela divulgação e a qualidade das informações apresentadas.

1.3 Organização do Texto

Este trabalho é organizado em uma introdução mais 5 capítulos, onde são abordados os seguintes itens:

Capítulo 2 - Referenciais Teóricos

Neste capítulo é introduzido os conceitos teóricos de tudo o que foi utilizado para desenvolver este trabalho. Possui tópicos sobre o sistema operacional Android, a API de conexão com a rede social Facebook, a linguagem Python e o framework Django.

Capítulo 3 - Metodologia e Desenvolvimento

Na metodologia deste trabalho é apresentado todo o processo de criação do aplicativo e do site de gerenciamento de eventos, além de como funciona a comunicação entre eles. Com muitos detalhes técnicos, este capítulo indica todas as classes utilizadas para implementar os recursos.

Capítulo 4 - Documentação

Neste capítulo é apresentado a documentação deste trabalho baseados em padrões de projeto da UML. Nesta documentação, está presente os requisitos de usuário, requisitos funcionais e não funcionais, diagramas de casos de uso e suas descrições.

Capítulo 5 - Estudo de Caso

Neste capítulo é apresentado os resultados de três estudos de caso aplicados à um ambiente simulado, onde foi possível comprovar a corretude do aplicativo e do site de gerenciamento. Os testes abordam a busca de eventos por GPS e por Cidade.

Capítulo 6 - Conclusão

Neste capítulo é apresentado as conclusões que foram tomadas à partir de todo o trabalho proposto. Conclusões sobre metodologia utilizada, a linguagem para criar a documentação do sistema e os resultados obtidos dos estudos de caso são exemplos de discussões impostas neste capítulo.

2 REFERENCIAIS TEÓRICOS

2.1 Android

Com o desenvolvimento de novas tecnologias para comunicação de dispositivos móveis, as informações podem ser acessadas onde quer que estejam, mesmo o usuário estando em movimento. Atualmente, as empresas buscam adaptar seus modelos de negócios para que possam oferecer seus produtos e serviços à partir de aplicativos móveis, aumentando suas vendas e o facilitando o atendimento ao cliente (LECHETA, 2013).

O Android é um sistema operacional móvel desenvolvido pela Google, que foi baseado no sistema operacional Linux. Possui diversas versões, sendo separadas por nível de *Application Programming Interface* (API). A primeira versão comercial do sistema foi a 1.6, chamada *Donut*. Com o passar do tempo, são lançadas novas versões do sistema operacional, entretanto, as empresas podem fazer suas próprias modificações no sistema, porque sua distribuição é livre, sob a licença *Apache* (LECHETA, 2013).

A Tabela 1 mostra a distribuição dos usuários do sistema operacional Android de acordo com a sua versão. A mudança de nível de API de uma versão para outra indica que novos recursos foram adicionados em relação à versão anterior. Apesar de possuir versões mais novas, a versão *KitKat* do Android é a mais utilizada atualmente.

Tabela 1 – Distribuição dos usuários por versão do Android

Versão	Nomes	API	Distribuição
2.2	Froyo	8	0.3%
2.3.3 - 2.3.7	Gingerbread	10	5.7%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.3%
4.1.x	Jelly Bean	16	15.6%
4.2.x		17	18.1%
4.3		18	5.5%
4.4	KitKat	19	39.8%
5.0	Lollipop	21	9.0%
5.1		22	0.7%

Fonte: (ANDROID DEVELOPERS, 2015a)

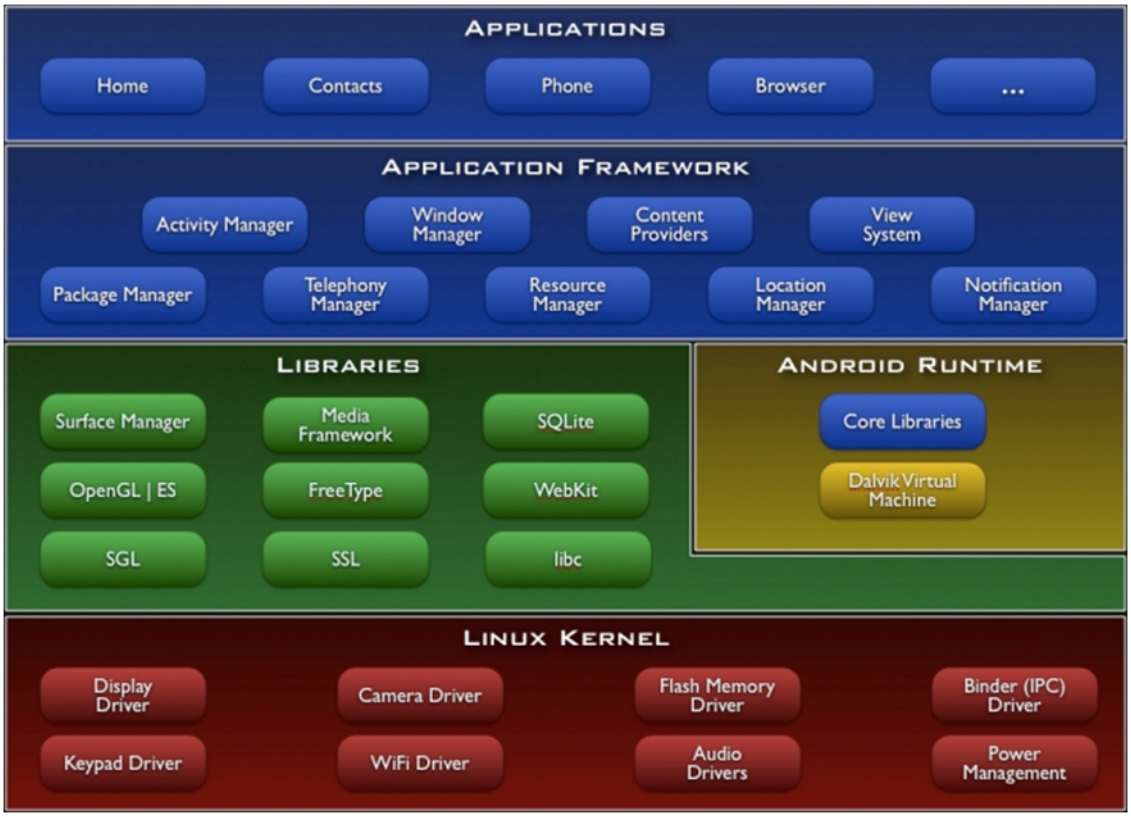
2.1.1 Arquitetura do sistema operacional Android

Para o sistema operacional Android funcionar corretamente em diversos modelos de aparelhos, sua arquitetura foi especialmente criada para que o desenvolvedor não se preocupe com o modelo do dispositivo que irá executar a aplicação. O sistema operacional

Android executa os aplicativos utilizando uma máquina virtual chamada *Dalvik Virtual Machine*, permitindo que todas as operações sejam tratadas igualmente pela máquina virtual, sendo repassado posteriormente para o sistema nativo (LECHETA, 2013).

A Figura 1 mostra a estrutura dos níveis de abstração propostos pelo sistema, e como funciona sua hierarquia e as relações entre elas.

Figura 1 – Arquitetura estrutural do sistema operacional Android



Fonte: (ARMAL, 2015)

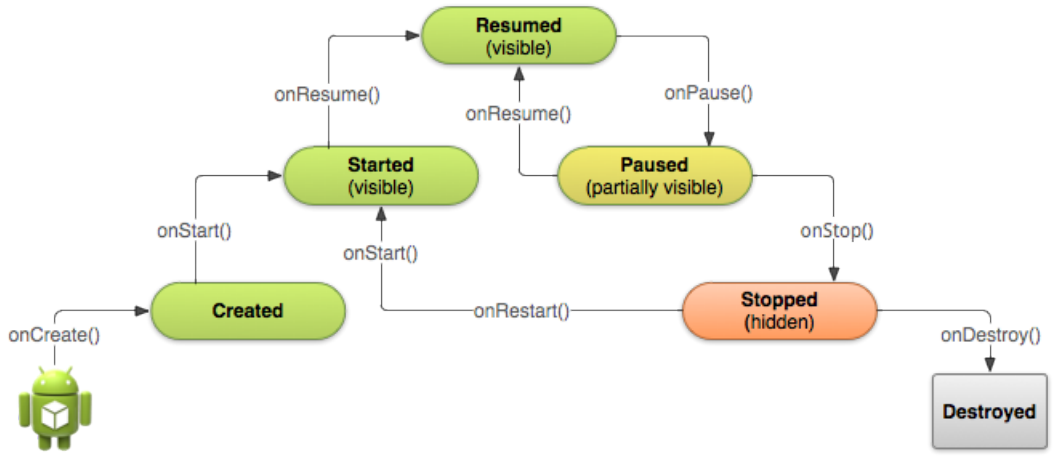
Na camada 1 da arquitetura está a execução de todos aplicativos, onde os aplicativos nativos do sistema e aqueles que são baixados pelo o usuário são executados. Na camada 2 da arquitetura estão os gerenciadores, onde são controlados os recursos específicos do aparelho, como o serviços de localização, notificações, telefonia, entre outros. Na camada 3 da arquitetura estão as bibliotecas e a base do ambiente virtual, que proporcionam o funcionamento correto das operações e dos recursos do aparelho. A camada 4 é a base do sistema Linux, onde estão localizados os controladores de dispositivos específicos de cada aparelho, permitindo o uso do teclado, câmera, caixas de som, internet sem fio e o controle de energia da bateria (ARMAL, 2015).

2.1.2 Activity e seu ciclo de vida

Uma *activity* no sistema operacional Android é uma classe que representa uma tela da aplicação. Ela possui métodos padrões que podem ser modificados para a customização de seu funcionamento. As *activitys* possuem um ciclo de vida, isto é, uma série de métodos que são chamados em uma ordem específica, que permite ao desenvolvedor iniciar ou desativar recursos que deseja no momento necessário (ANDROID DEVELOPERS, 2015c).

A Figura 2 mostra como funciona o ciclo de vida de uma *activity*, e qual a ordem dos métodos chamados durante toda a sua execução.

Figura 2 – Ciclo de vida de uma activity



Fonte: (ANDROID DEVELOPERS, 2015c)

Ao iniciar a *activity*, o método *onCreate()* é executado. Este método fica responsável por desenhar a tela da aplicação, colocando textos, botões e imagens em seus lugares corretos. Seguindo o ciclo, são executados o métodos *onStart()* e *onResume()*, que são utilizados para iniciar os serviços necessários no aplicativo, como por exemplo o sistema de localização e o inicio da requisição dos eventos para o servidor. Como estes serviços não fazem parte do desenho da tela, eles são executados nestes métodos.

Durante a execução da *activity*, ela pode ser interrompida por diversos motivos: o aparelho pode receber uma ligação, exibição de uma caixa de alerta, o usuário pode sair da aplicação, etc. Portanto, quando a *activity* é interrompida, são executados os métodos *onPause()* e *onStop()*, destinados para desativar os serviços utilizados na tela. Finalmente, o método *onDestroy()* é executado para destruir a tela e liberar a memória consumida para o sistema.

É fundamental para o desenvolvedor entender como funciona este ciclo de vida, para que os recursos do aparelho utilizados em sua aplicação sejam usados de forma correta, para que não haja desperdício de processamento, memória e energia.

2.1.3 Location API

Um recurso que as aplicações móveis no Android possuem é a possibilidade de conhecer a localização de seu usuário. Isso permite que as aplicações possam ser mais dinâmicas, mostrando seu conteúdo de acordo com o local do usuário, por exemplo. Na biblioteca *Google Play Services*, que está disponível dentro do *Android Software Development Kit* (SDK), existe o pacote Location API que permite ao desenvolvedor obter estes dados de localização, em latitude e longitude ou convertê-lo em um endereço (ANDROID DEVELOPERS, 2015b).

Com o pacote Location API, é possível:

a) Obter o último local conhecido do aparelho, de modo imediato;

b) Receber atualizações do posicionamento, em tempo real;

c) Conhecer o possível endereço de uma coordenada geográfica.

O sistema operacional Android verifica a disponibilidade de cada um dos provedores de localização que ele possui, sendo eles o receptor de coordenadas geográficas a partir do *Global Positioning System* (GPS), a rede sem fio ou a rede de telefonia móvel. É escolhida a localização do provedor que retornar a melhor precisão, e então este dado é repassado para todas as aplicações que estejam solicitando esta informação no momento (ANDROID DEVELOPERS, 2015b).

2.1.4 Facebook SDK

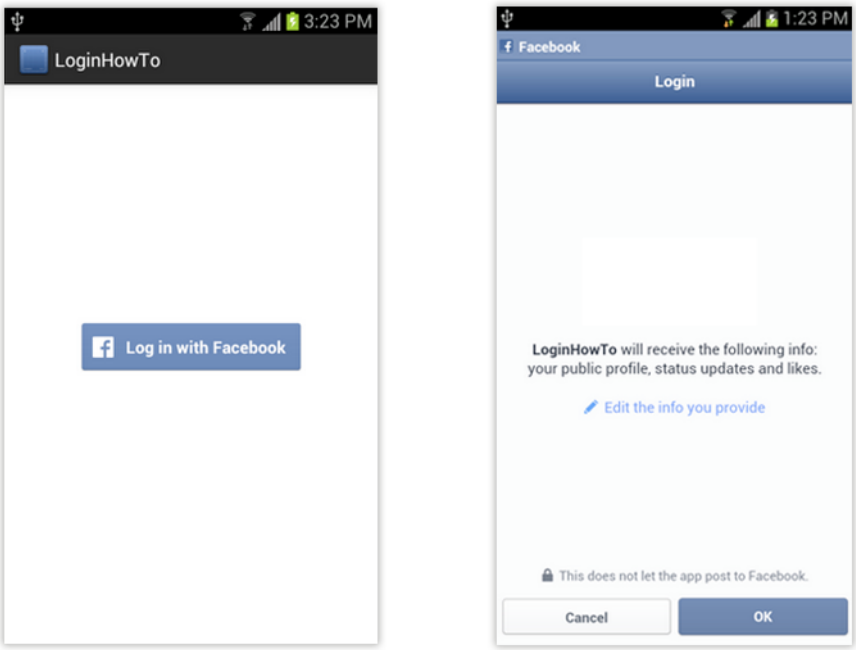
A rede social *Facebook* criou uma biblioteca para o sistema operacional Android que permite um acesso facilitado aos dados de seus usuários. Os dados fornecidos pela biblioteca dependem de permissões, garantindo assim a privacidade de seus usuários. Em sua documentação oficial[1] são apresentadas todas as classes e interfaces disponíveis para uso e como utilizá-las (FACEBOOK, 2015b).

Com esta biblioteca, é possível criar telas que possibilitam o login de um usuário, o compartilhamento de publicações, a criação de eventos, enviar convites para o acesso ao aplicativo e obter outras informações adicionais, como lista de amigos, lista de páginas que o usuário curtiu, álbum de fotos, por exemplo (FACEBOOK, 2015b).

Na Figura 3, é apresentado o botão que permite o acesso de um usuário dentro da aplicação, gerado pela classe *LoginButton* que está dentro da biblioteca do Facebook SDK. Ao tocar neste botão, é apresentado ao usuário todas as permissões que a aplicação deseja obter de sua conta do Facebook. Logo abaixo, há dois botões que dão a possibilidade de aceitar as condições propostas ou cancelar a operação (FACEBOOK, 2015a).

[1] https://developers.facebook.com/docs/reference/android/current

Figura 3 – Uso do botão de Login do Facebook SDK



Fonte: (FACEBOOK, 2015a)

Após a conclusão do login de um usuário, a biblioteca retorna um código alfanumérico chamado *accessToken*. Este código é necessário para validar todas as requisições de dados feitas pela aplicação, evitando a necessidade de fazer o login para cada solicitação realizada. Todas as permissões aceitas pelo usuário são salvas pelo Facebook e ficarão disponíveis na lista de aplicativos permitidos do usuário, porém essas permissões podem ser revogadas pelo usuário à qualquer momento (FACEBOOK, 2015a).

2.2 Python

Python é uma linguagem de programação orientada a objetos. O código-fonte em Python é executado à partir de um interpretador, ou seja, cada linha de código é executada separadamente, diferente de uma linguagem que utiliza um compilador, que lê todo o conteúdo do código-fonte para então gerar um código-objeto (LUTZ; ASCHER, 2007).

O interpretador e a documentação completa do Python podem ser encontrados em seu site oficial[2], para as versões 2 e 3. Portanto, a linguagem Python é considerada rápida, poderosa e fácil de aprender, podendo ser utilizada para realizar diversas tarefas em geral, como ser apenas um componente de um projeto de software ou um programa independente completo (PYTHON SOFTWARE FOUNDATION, 2015).

É possível também utilizar a linguagem Python para criar servidores Web. No

[2] https://www.python.org/downloads/

código-fonte do Python, é utilizado sockets de conexão para realizar a comunicação entre os clientes e o servidor, enviar e receber e-mails, realizar busca em páginas já existentes na internet, retornar páginas feitas em *HyperText Markup Language* (HTML), entre outras funções (LUTZ; ASCHER, 2007).

Para estruturar os objetos do Python, foi utilizado o módulo "json", que está presente por padrão no Python 2.7.

O *JavaScript Object Notation* (JSON) é um formato de estruturação de dados, baseado na linguage JavaScript. Utiliza convenções de linguagens convencionais como C, C++, Java, JavaScript, entre outras. É definida como uma lista de objetos ou objetos com um ou mais pares de nome/valor (JSON, 2015).

2.3 Django

O Django é um *framework* de código aberto voltado para o desenvolvimento ágil de sistemas Web. Feito em linguagem Python, permite a construção de aplicações de alta qualidade para a internet, cuidando de grande parte do desenvolvimento repetitivo, utilizando o padrão de arquitetura de software chamado MVC (DJANGO, 2015b).

A necessidade de se obter grandes modificações em projetos de curto espaço de tempo levou um grupo de desenvolvedores a criar esta ferramenta. Os desenvolvedores Adrian Holovaty e Simon Willison, criaram a primeira versão que facilitava novas implementações e permitia realizar modificações ágeis nos sistemas feitos para a internet (HOLOVATY; KAPLAN-MOSS, 2007).

2.3.1 Estrutura dos projetos e aplicações Django

Neste *framework*, os sistemas web são categorizados em projetos e aplicações.

Um projeto é uma coleção de configurações para uma instância de Django, incluindo as configurações de acesso ao banco de dados, configurações específicas do Django e configurações específicas de aplicações. Um único projeto pode conter várias aplicações, e essas aplicações podem relacionar-se entre si (HOLOVATY; KAPLAN-MOSS, 2007).

Uma aplicação dentro do Django é uma parte da implementação que pode ou não ser independente do projeto que está sendo utilizado. Portanto, uma aplicação Django pode ser aplicada em vários projetos por não existir dependência de dados, facilitando o reúso de código (HOLOVATY; KAPLAN-MOSS, 2007).

2.3.2 A arquitetura Model-View-Controller

O Django utiliza o padrão de desenvolvimento *Model-View-Controller*. De uma maneira geral esta arquitetura separa os códigos-fonte pelas suas funções dentro do um projeto, fazendo com que sua implementação e manutenção sejam mais rápidas e de fácil compreensão (SROKA, 2015).

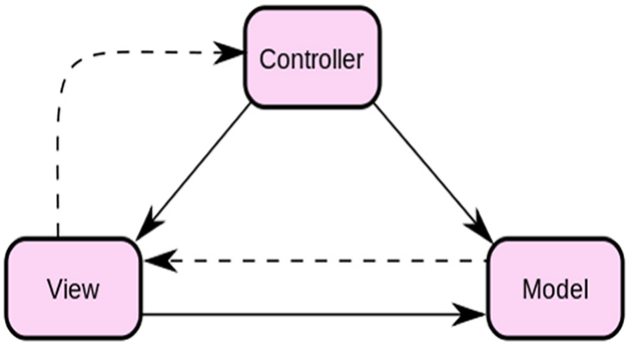
O Controller tem a função de receber a requisição vinda do servidor web, verificando seu conteúdo e encaminhando a requisição para a View correta que irá tratá-la (SROKA, 2015).

A View recebe a requisição do Controller e faz todo o processamento necessário para gerar o conteúdo que será exibido. Se for necessário realizar uma consulta no banco de dados, a View faz a solicitação para o Model, pois apenas ele tem acesso direto ao banco de dados (SROKA, 2015).

O Model recebe a solicitação e cria a consulta e faz o acesso ao banco de dados do sistema. Ao receber o resultado, o Model encapsula os dados e os envia para a View. Ao terminar todo seu processamento, a View encaminha sua resposta para o Controller, que então irá despachar a resposta para o cliente à partir do servidor web (SROKA, 2015).

A Figura 4 mostra como funciona a relação entre os objetos envolvidos nessa arquitetura. A linha sólida indica um fluxo de solicitação de dados, enquanto a linha tracejada indica o fluxo de resposta de dados.

Figura 4 – Arquitetura Model-View-Controller



Fonte: (SROKA, 2015)

Esta arquitetura possibilita aos desenvolvedores a separação do trabalho, sem a criação de dependências entre as partes do código. Por exemplo, o desenvolvedor pode realizar a mudança da URL sem afetar outras partes implementadas. O designer gráfico do site pode realizar modificações na página HTML sem que seja necessário modificar o código-fonte que o gera. O administrador do banco de dados do sistema tem a liberdade de modificar os nomes dos campos da tabela sem que seja necessário a modificação das requisições feitas pelo sistema (HOLOVATY; KAPLAN-MOSS, 2007).

2.3.3 PostgreSQL e PostGIS

O PostgreSQL é um banco de dados relacional que foi criado à partir de uma derivação do pacote postgres, escrito na Universidade da Califórnia. Proposto pelo professor Michael Stonebraker, o Postgres passou por várias versões, até que a primeira versão do sistema foi exibida em 1988 (SOURCEFORGE, 2015).

No meio da década de 90, foi adicionado o interpretador de linguagem SQL ao postgres, que após isso passou a ser chamado de Postgres95. Sua nova versão foi escrito na linguagem ANSI C e era quase duas vezes mais rápido que sua versão anterior (SOURCEFORGE, 2015).

2.3.4 GeoDjango

O GeoDjango é um módulo incluso dentro do Django que permite criar aplicações baseados em dados geográficos. Neste módulo é possível armazenar pontos usando os valores de latitude e longitude, armazenar áreas usando valores de vários pontos e fazer consultas no banco de dados utilizando os dados geográficos e modelos geométricos (DJANGO, 2015a).

As consultas espaciais dentro do GeoDjango é feito pelo módulo *GeoQuerySet*, que de acordo com o banco de dados que for utilizado no projeto, os métodos de consultas espaciais ficam disponíveis (ANDERSON, 2015).

De acordo com a Figura 5, as extensões de bancos de dados suportados são PostGIS, Oracle e SpatiaLite. O Oracle é o menos suportado entre os três, além de ser necessário comprar sua licença de uso. O SpatiaLite suporta grande número de métodos, entretanto sua arquitetura o torna lento em comparação ao PostGIS. O PostGIS é o único que suporta todos os métodos e seu uso é livre sobre a *General Public License*[3] (ANDERSON, 2015).

Utilizando as tecnologias atuais que existem para banco de dados espaciais, o *GeoQuerySet* recebe os comandos do programador e os transforma em consultas dentro do banco de dados. Neste tipo de banco de dados é possível utilizar outros tipos de filtros para os pontos e áreas, como por exemplo **near** (perto), **contains** (contém), **out** (fora), entre outros filtros (DJANGO, 2015a).

[3] http://opensource.org/licenses/gpl-2.0.php

Figura 5 – Relação entre o banco de dados e os métodos de consulta geográficos

Method	PostGIS	Oracle	SpatiaLite
GeoQuerySet.area()	X	X	X
GeoQuerySet.centroid()	X	X	X
GeoQuerySet.collect()	X		
GeoQuerySet.difference()	X	X	X
GeoQuerySet.distance()	X	X	X
GeoQuerySet.envelope()	X		X
GeoQuerySet.extent()	X	X	
GeoQuerySet.extent3d()	X		
GeoQuerySet.force_rhr()	X		
GeoQuerySet.geohash()	X		
GeoQuerySet.geojson()	X		X
GeoQuerySet.gml()	X	X	X
GeoQuerySet.intersection()	X	X	X
GeoQuerySet.kml()	X		X
GeoQuerySet.length()	X	X	X
GeoQuerySet.make_line()	X		
GeoQuerySet.mem_size()	X		
GeoQuerySet.num_geom()	X	X	X
GeoQuerySet.num_points()	X	X	X
GeoQuerySet.perimeter()	X	X	
GeoQuerySet.point_on_surface()	X	X	X
GeoQuerySet.reverse_geom()	X	X	
GeoQuerySet.scale()	X		X
GeoQuerySet.snap_to_grid()	X		
GeoQuerySet.svg()	X		X
GeoQuerySet.sym_difference()	X	X	X
GeoQuerySet.transform()	X	X	X
GeoQuerySet.translate()	X		X
GeoQuerySet.union()	X	X	X
GeoQuerySet.unionagg()	X	X	X

Fonte: (ANDERSON, 2015)

3 METODOLOGIA E DESENVOLVIMENTO

Para o desenvolvimento da aplicação móvel, foi utilizado o Android Studio, que é a *Integrated Development Environment* (IDE) oficial de desenvolvimento de aplicações para o sistema operacional Android. Este programa é encontrado em seu site oficial[1] e sua distribuição é gratuita, sobre a licença Apache[2].

Junto com o Android Studio, há um pacote de bibliotecas de desenvolvimento chamado Android SDK, que contém todas as bibliotecas e códigos necessários para o funcionamento da ferramenta, bem como os sistemas de simulação e de compilação. Dentro da IDE de desenvolvimento, foi utilizada a linguagem Java em conjunto com as bibliotecas do Android SDK. Para a localização do aparelho, foi utilizada a Location API do Android SDK, que proporciona a localização de acordo com o melhor provedor disponível. Para o cadastro dos usuários na aplicação, foi utilizado Facebook API, onde são obtidos os dados necessários para registro e então enviados para o servidor.

Junto ao desenvolvimento da aplicação móvel, foi desenvolvida a página de gerenciamento de eventos, que é um site na internet onde os promotores poderão cadastrar seus eventos para que sejam apresentados na aplicação. Foi feita em duas fases: Criação da página de apresentação e a página de painel de controle dos eventos.

A página de gerenciamento de eventos foi desenvolvida em HTML e Python, utilizando o *framework* de desenvolvimento Django. Este *framework* possibilita o desenvolvimento ágil do projeto da página, que por sua vez foi também o servidor da aplicação móvel, que tratou todas as solicitações de dados feitas pela aplicação.

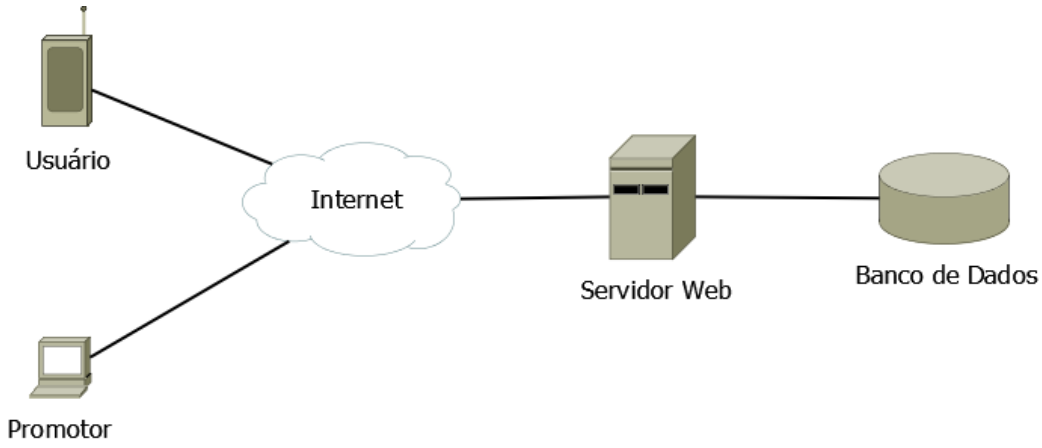
3.1 Arquitetura geral do sistema

Para que o site de gerenciamento de eventos e a aplicação possam se comunicar, é necessário que haja um servidor que responda à todas as requisições realizadas. Este servidor contém o banco de dados onde todos os eventos presentes no sistema estão armazenados. Os dados específicos desta comunicação pode ser visto na documentação, descrito no Capítulo 4.

A Figura 6 mostra a arquitetura do sistema. Há um banco de dados PostgreSQL que possui comunicação direta com um servidor web. Este servidor, feito em Django, trata as requisições do aplicativo e também as requisições do site de gerenciamento.

[1] http://developer.android.com/sdk/index.html
[2] http://www.apache.org/licenses/LICENSE-2.0

Figura 6 – Arquitetura geral do sistema



Fonte: Elaborada pelo autor.

3.2 Aplicação móvel

As interfaces de usuário nas aplicações Android podem ser diagramadas pelo próprio Android Studio, que ao serem salvas são convertidas para arquivos de layout. Dentro dos arquivos de layout, que é escrito na linguagem *eXtensible Markup Language* (XML), estão as definições dos objetos que estarão presentes na tela, junto com suas configurações de margens e alinhamentos. Entretanto, é possível criar toda a interface dentro do código-fonte da *activity*, porém nos arquivos de layout é mais simples criar as relações de posições e de configurações de cada objeto da tela (FIORINI, 2015).

A Figura 7 apresenta um exemplo de arquivo de layout utilizado no sistema Android. Neste exemplo, há uma tag chamada *RelativeLayout*, que representa a própria classe *RelativeLayout* do Android. Dentro desta tag há mais três tags, sendo elas a *include*, que inclui um arquivo de layout externo, a *ImageView* que exibe uma imagem, e um *TextView*, que exibe um texto simples na tela. Dentro do código-fonte da *activity*, no método *onCreate()*, este arquivo é referenciado com o método *setContentView()*.

Junto com os layouts, existem as Atividades (*activitys*), que é o ambiente no Android que desenha o layout na tela do aparelho e possui todo o código-fonte necessário para o funcionamento da tela. Portanto, cada tela da aplicação irá possuir um arquivo com as classes da linguagem Java que será a atividade, junto com um arquivo XML que será o layout da tela.

3.2.1 Tela principal

Esta tela é a *activity* principal da aplicação. Após o registro do aparelho no servidor de dados feita pela tela de acesso (apresentada na subseção 3.2.3), esta tela se torna a primeira a ser executada quando a aplicação é iniciada.

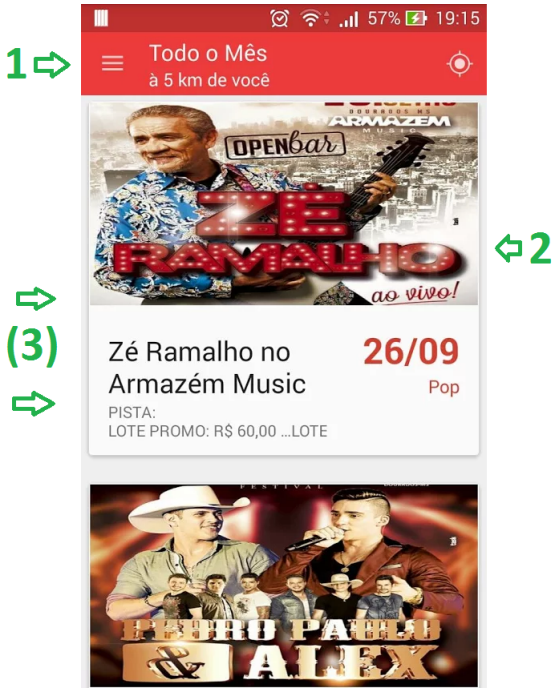
Figura 7 – Exemplo de um arquivo XML de layout

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.biasi.party.Principal">
    <include
        android:id="@+id/inc_toolbar_top"
        layout="@layout/toolbar_top"/>
    <ImageView
        android:layout_width="match_parent"
        android:adjustViewBounds="true"
        android:contentDescription="@null"
        android:scaleType="fitXY"
        android:layout_height="wrap_content"
        android:layout_below="@+id/inc_toolbar_top"
        android:id="@+id/festa_image" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Título da Festa"
        android:id="@+id/text1"
        android:layout_below="@+id/festa_image"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp" />
</RelativeLayout>
```

Fonte: Elaborada pelo autor.

Ela é composta de uma barra de ferramentas posicionada no topo da tela (1), uma lista de eventos localizada no centro (2) e um menu oculto no lado esquerdo da tela (3), como apresentado na Figura 8. Cada um desses itens serão explicados separadamente nas seções abaixo.

Figura 8 – Escopo da tela principal



Fonte: Elaborada pelo autor.

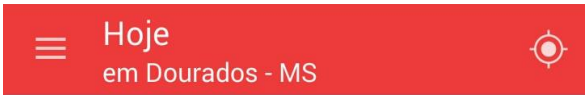
3.2.1.1 Barra de ferramentas

A barra de ferramentas da tela principal fica localizada no topo da tela e é implementada utilizando a classe *Toolbar*. Ela é composta de um ícone de três barras horizontais orientado à esquerda que possui a função de mostrar o menu lateral.

Ao lado do ícone há duas linhas de texto, sendo que a primeira mostra o intervalo de datas dos eventos apresentados, sendo possível os textos "Hoje", "Próximos 7 dias" ou "Todo o mês". Na segunda linha, há a indicação do modo de busca de eventos, exibindo o nome da cidade ou a distância máxima de um evento.

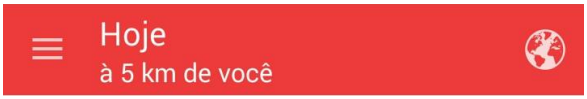
Orientado à direta da barra de ferramentas, há um ícone de um globo ou de localização, onde é possível alternar entre os modos de busca por GPS ou busca por cidade. A Figura 9 e a Figura 10 apresenta a diferença da barra de ferramentas entre os modos de busca e comportamento do ícone de acordo com o modo de busca ativado.

Figura 9 – Busca por Cidade



Fonte: Elaborada pelo autor.

Figura 10 – Busca por GPS



Fonte: Elaborada pelo autor.

3.2.1.2 Lista de eventos

Abaixo da barra de ferramentas está a lista de eventos, criada utilizando a classe *RecyclerView*. Esta classe carrega na memória do aparelho apenas os itens da lista que estão visíveis na tela no momento, diminuindo o consumo de memória RAM. O item (2) presente na Figura 8 mostra detalhadamente um exemplo de um evento na lista.

Baseado em um cartão, o item possui uma imagem no topo que é enviada pelo promotor do evento. Logo abaixo da imagem, há o título do evento e as duas primeiras linhas de descrição. Orientado à direita, está a data do evento apresentado em dia e mês e o estilo musical do evento.

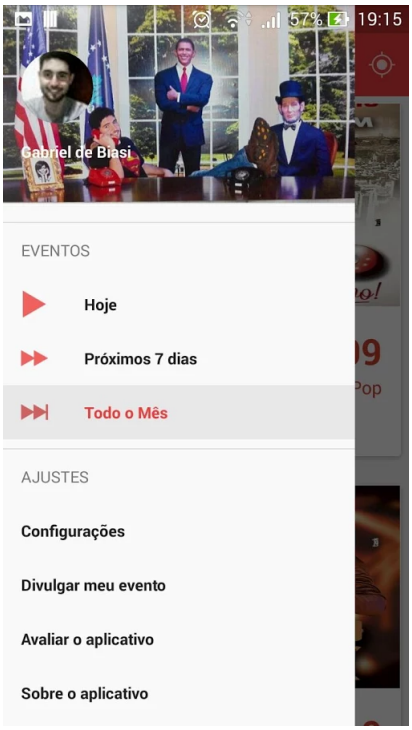
3.2.1.3 Menu lateral

No lado esquerdo da *activity* principal, há um menu lateral implementado utilizando a classe *DrawerLayout*. Neste menu, há um cabeçalho que mostra o nome do usuário, sua

foto de perfil e sua foto de capa ao fundo. Todos estes dados de usuário são obtidos pelo Facebook API.

O usuário pode alternar entre as datas dos eventos apresentados, podendo visualizar entre os eventos que irão acontecer no dia, nos próximos 7 dias ou durante todo o mês. No final do menu, há uma opção de acesso à *activity* de configurações. A Figura 11 mostra o escopo do menu lateral da aplicação.

Figura 11 – Escopo da tela principal com o menu lateral aberto



Fonte: Elaborada pelo autor.

3.2.2 Tela de visualização de eventos

Quando o usuário toca em um evento presente na lista, ele é direcionado para a tela de visualização. Esta tela é construída à partir da classe *CoordinatorLayout*, que controla a visualização dos itens que são exibidos na tela. A tela é composta de uma imagem no topo da tela onde aparece o banner junto ao título do evento, seguido de uma lista de informações do evento. As informações estão especificadas na lista abaixo:

a) Data e Hora

b) Estilo

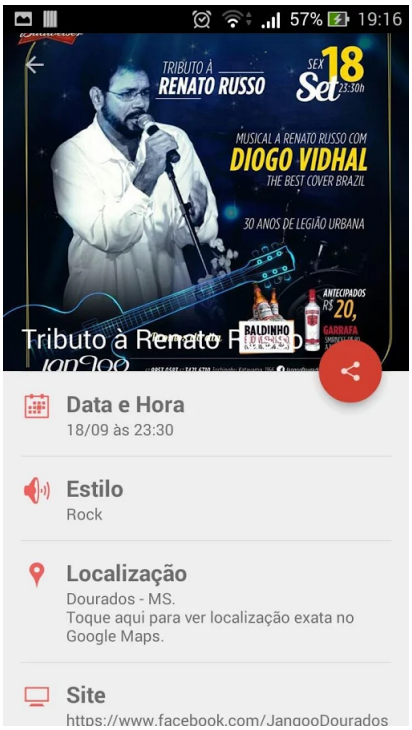
c) Localização

d) Site

e) Descrição do Evento

Ao tocar no item **Site**, o usuário é direcionado para o navegador de internet padrão para acessar o site do evento que foi especificado anteriormente pelo promotor. Ao tocar no item **Localização**, o usuário é direcionado ao aplicativo de mapas padrão do aparelho, mostrando a posição exata do evento. O escopo desta tela pode ser visualizado na Figura 12.

Figura 12 – Escopo da tela de visualização de eventos



Fonte: Elaborada pelo autor.

3.2.3 Tela de acesso

Ao iniciar o aplicativo pela primeira vez, será apresentado ao usuário uma tela informando que é preciso entrar com seus dados de acesso do Facebook para usar o aplicativo. Esta *activity* utiliza uma classe chamada *LoginButton*, oferecido pelo Facebook API, que coloca um botão no centro da tela que possibilita o usuário entrar com seus dados, indicado pelo item (1) na Figura 13.

Após o retorno dos dados pelo Facebook API, é iniciado o processo de registro do usuário e de seu aparelho no servidor de dados da aplicação. Os seguintes campos são enviados para o servidor de dados:

a) ID do usuário no Facebook;

b) Nome completo do usuário no Facebook;

c) ID no aparelho.

Figura 13 – Escopo da tela de acesso

Fonte: Elaborada pelo autor.

Ao receber estes dados, o servidor envia uma *flag* de confirmação para o aplicativo indicando que o registro foi efetuado com sucesso e está pronto para receber requisições de eventos deste aparelho.

3.2.4 Tela de configurações

A tela de configurações é onde o usuário definirá todas suas preferências dentro da aplicação. Ela é acessada no item "Configurações" que está no menu lateral da tela principal. Esta *activity* é implementada utilizando a classe *PreferenceActivity*, que gera automaticamente uma tela de configurações onde é possível ver e modificar os valores de preferências, como apresentado na Figura 14.

Esta tela possui os seguintes itens de configuração:

a) Filtro por estilo musical (Lista)

b) Alternar entre busca por GPS ou busca por cidade (Caixa de seleção)

c) Alcance da Busca (Valor numérico)

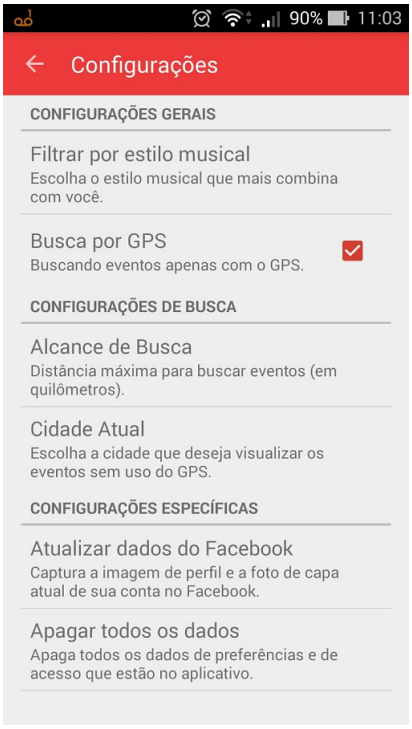
d) Cidade Atual (Lista)

e) Atualizar dados do Facebook (Ação)

f) Apagar todos os dados (Ação)

A classe *PreferenceActivity* cria um arquivo oculto no sistema do aparelho do usuário, onde todos os dados definidos nesta tela são salvos para que possam ser consultados

Figura 14 – Escopo da tela de configurações



Fonte: Elaborada pelo autor.

de qualquer *activity* da aplicação posteriormente. É utilizado geralmente para registrar pequenos valores, onde não é necessário criar um banco de dados para armazená-los.

3.3 Site de gerenciamento de eventos

Para que os eventos possam ser exibidos na aplicação, é necessário que um promotor de eventos tenha uma interface onde através de um painel de controle seja possível fazer o gerenciamento de seus eventos.

Portanto, o site de gerenciamento de eventos será um site na internet que permite ao promotor criar eventos, modificar informações ou excluir eventos dentro do sistema. Para isto, será necessário que o promotor seja cadastrado por um administrador do sistema.

O site é hospedado junto ao servidor de dados de eventos do aplicativo, para que a comunicação seja mais rápida e eficiente. Na seção 3.4, o servidor de dados é explicado com mais detalhes.

3.3.1 Página de apresentação

A página de apresentação possui um resumo dos recursos que o aplicativo possui. Logo abaixo desta seção da página há um área para realizar o acesso para o painel de controle de eventos. A Figura 15 mostra o topo da página de apresentação.

Figura 15 – Página de apresentação do aplicativo

Fonte: Elaborada pelo autor.

3.3.2 Gerenciamento de eventos

O site de gerenciamento de eventos mostra todos os eventos do promotor, além de poder criar, modificar ou excluir eventos.

Na Figura 16, está o esquema do painel de controle. No topo, há um menu de acesso as áreas do site, contendo um link para a lista de eventos do promotor (item 1), um link para a página de registro de eventos (item 2) e um link de saída do painel de controle (item 3).

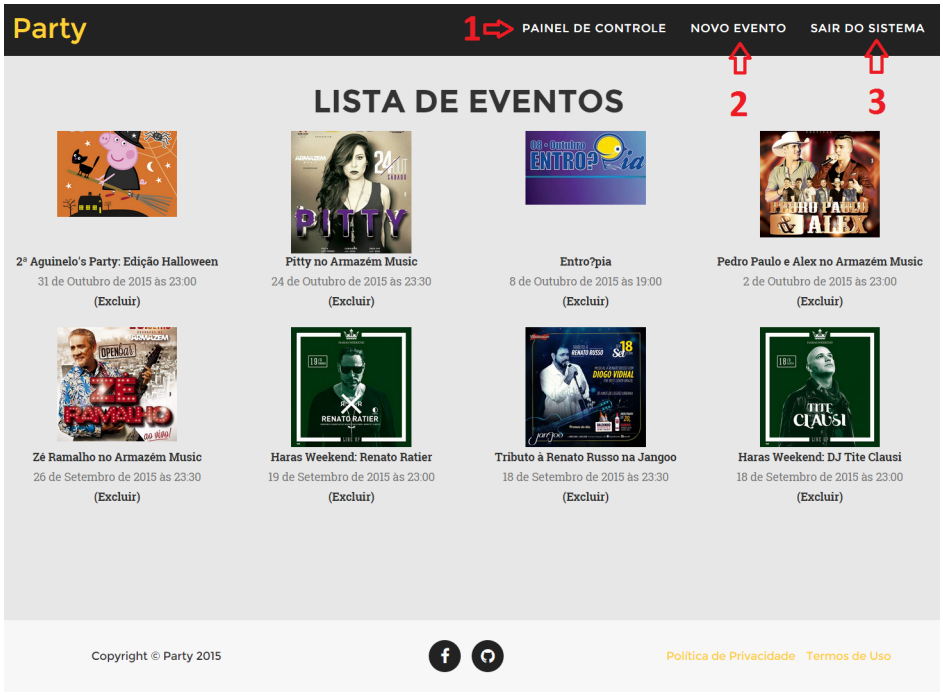
Abaixo do menu de acesso, fica a lista de eventos do promotor. Nesta lista, cada item representa um evento composto pelo banner, junto ao título do evento e sua data. Ao clicar sobre o banner, o promotor é redirecionado para uma página onde ele pode modificar os dados que desejar.

3.4 Comunicação entre a aplicação móvel e o servidor

Para que a aplicação móvel obtenha os dados de eventos do servidor, há uma interface própria de comunicação. Através de uma requisição feita pelo o aplicativo, o servidor irá fazer uma consulta no banco de dados que então irá retornar uma lista de eventos para que a aplicação possa exibir na tela do usuário.

O gerenciador de banco de dados utilizado neste trabalho foi o PostgreSQL, junto à uma extensão geográfica chamada PostGIS.Para realizar as consultas geográficas no

Figura 16 – Página do painel de controle

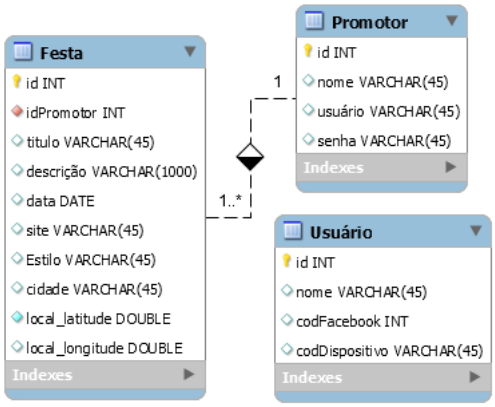


Fonte: Elaborada pelo autor.

banco de dados PostgreSQL, foi utilizado o *middleware* GeoDjango, que realiza a consulta por meio de variáveis geográficas. O uso do GeoDjango neste trabalho pode ser visualizado nas linhas 60 e 66, no código-fonte do Apêndice A.

O diagrama da Figura 17 mostra a estrutura de entidades e relacionamentos proposto inicialmente no sistema, que foi utilizado como referência para criar o banco de dados.

Figura 17 – Diagrama Entidade-Relacionamento



Fonte: Elaborada pelo autor.

A requisição acontece de acordo com o diagrama da Figura 18. Esta comunicação é

realizada por uma conexão HTTP, os dados são enviados no cabeçalho da mensagem e os dados recebidos do servidor vem formatados em JSON.

Figura 18 – Diagrama da requisição feita pela aplicação móvel

Fonte: Elaborada pelo autor.

No passo 1, o usuário selecionou uma opção que dispara a ação de requisição de eventos dentro do aplicativo. Então, o aplicativo reúne todos os campos necessários para enviar a requisição para o servidor de dados. Estes campos são:

a) ID do Usuário (Para validar a requisição);

b) Categoria do Evento (Buscar por categoria selecionada);

c) Cidade Atual (Eventos da cidade selecionada, pode ser nulo);

d) Latitude e Longitude (Se a cidade é nula, os dados de localização são utilizados);

e) Distância (Distância máxima para buscar eventos à partir do local do usuário).

No passo 2, o servidor recebe a requisição e realiza a análise dos dados, verificando sua autenticidade buscando o ID do usuário no banco de dados e os campos são verificados em busca de valores inválidos. Após a análise, a consulta por eventos no banco de dados é feita pelo servidor.

No passo 3, ao receber a resposta do banco de dados, o servidor estrutura os dados da resposta em formato JSON para serem enviados ao aplicativo.

Caso seja encontrado algum campo inválido durante a análise feita pelo servidor, a resposta do servidor será um erro HTTP com código 404. Caso não haja eventos de acordo com os filtros aplicados pelo usuário, um objeto JSON vazio é retornado.

No passo 4, ao receber a resposta do servidor, o aparelho salva cada um dos eventos presentes na lista de objetos JSON em um banco de dados interno e busca por possíveis inconsistências entre os valores atuais do banco e os novos valores recebidos pelo servidor. Por exemplo, caso a requisição envie um mesmo evento que esteja presente no banco de dados do aparelho, os dados são apenas atualizados.

Para informações mais completas sobre como o servidor trata as informações e realiza a busca por eventos, veja todo o código-fonte localizado no Apêndice A que foi utilizado no servidor para realizar esta tarefa.

O capítulo seguinte apresenta um escopo de uma documentação aplicada sobre este sistema. Várias definições usadas neste capítulo são apresentadas nesta documentação.

4 DOCUMENTAÇÃO

Este capítulo apresenta uma formatação da documentação do aplicativo desenvolvido e do site de gerenciamento utilizando conceitos da *Unified Model Language* (UML).

A UML é uma linguagem utilizada para criar documentações, visualizações e especificações de sistemas de software. Baseada em modelos predefinidos, a UML faz com que todo o processo de desenvolvimento de software seja feita de maneira correta e organizada (BOOCH; RUMBAUGH; JACOBSON, 2006).

Por meio de seus variados tipos de diagramas, é possível representar sistemas de softwares sob diversos meios de visualização. Facilitando a comunicação de todas as pessoas envolvidas no processo de desenvolvimento de um sistema (gerentes de projeto, analistas, programadores, testadores) por apresentar um estilo de linguagem de fácil aprendizagem (BOOCH; RUMBAUGH; JACOBSON, 2006).

Esta documentação está estruturada primeiramente nas definições dos requisitos do usuário, onde os requisitos mostram uma visão abstrata do cliente à ser passada ao engenheiro de software. Após isso, os requisitos do sistema são criados baseados nos requisitos do usuário. Os requisitos do sistema mostram uma ideia mais aprofundada sobre o sistema, utilizando uma linguagem mais técnica.

O último assunto descrito neste capítulo é a visão dos casos de uso, onde os atores do sistema e o diagrama do modelo de casos de uso são apresentados, junto ao diagrama de atividades.

4.1 Definição dos requisitos do usuário

Nesta seção, foram definidos os requisitos do usuário. Os requisitos do usuário descrevem as funções e restrições do sistema de uma forma abstrata, que indica o ponto de vista do dono do sistema.

RU1 O administrador pode registrar promotores no sistema à qualquer momento.

RU2 O promotor pode registrar eventos no sistema à qualquer momento.

RU3 O usuário deve ser capaz de procurar por eventos que estejam próximos à ele.

RU4 O usuário deve ser capaz de procurar por eventos de uma cidade desejada.

4.2 Especificação dos requisitos de sistema

Nesta seção estão descritos os requisitos de sistema. Os requisitos de sistema devem ser padronizados e mais ricos em detalhes em comparação aos requisitos do usuário. Estão divididos entre requisitos funcionais e não funcionais.

4.2.1 Requisitos funcionais

RF1 O sistema deve manter um cadastro de todos os promotores de eventos. Os dados de um promotor são: Nome, Usuário e Senha.

RF2 O sistema deve manter um cadastro de todos os eventos criados pelos promotores. Os dados de um evento são: Identificador do Promotor, Título, Descrição, Data, Site, Banner, Cidade, Localização e Estilo.

RF3 O sistema deve manter um cadastro de todos os usuários que instalarem o aplicativo em seus dispositivos. Os dados de um usuário são: Nome, Identificador do Facebook e Modelo do Dispositivo.

RF4 O sistema deve disponibilizar uma interface de registro de eventos para os promotores. (Requisitos Relacionados: RF1, RF2)

RF5 O sistema deve disponibilizar uma interface de busca e visualização de eventos para os usuários. (Requisitos Relacionados: RF2, RF3)

RF6 O sistema deve permitir que o usuário altere suas preferências de buscas de eventos. (Requisitos Relacionados: RF2, RF3)

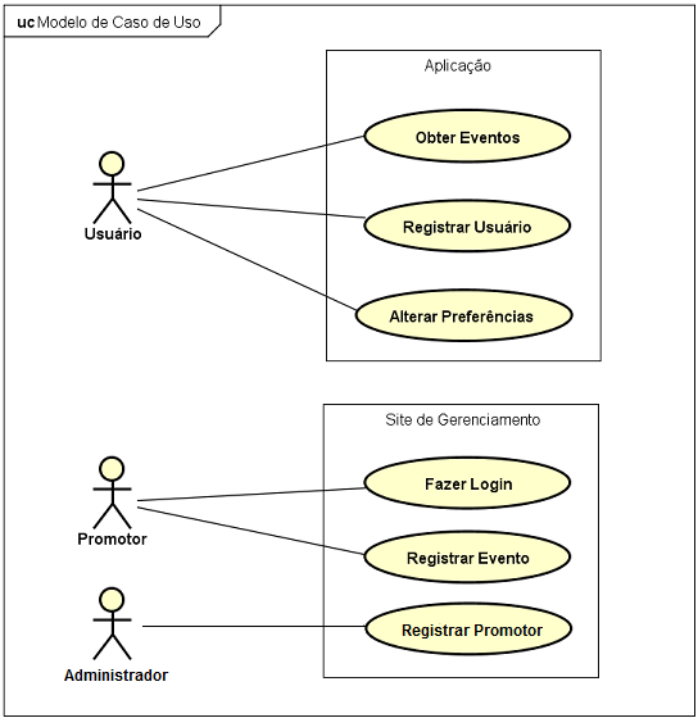
4.2.2 Requisitos não funcionais

RNF1 O sistema deve remover do banco de dados os eventos que já foram realizados. (Requisitos Relacionados: RF2)

4.3 Visão de casos de uso

Esta seção apresenta o diagrama de casos de uso deste trabalho, uma breve introdução sobre os atores presentes nesta documentação e o diagrama de atividade sobre o caso de uso “Obter Eventos”. A Figura 19 mostra como está estruturado os casos de uso e suas ligações com os atores.

Figura 19 – Diagrama do modelo de casos de uso



Fonte: Elaborada pelo autor.

4.3.1 Descrição dos atores

Estes são os atores presentes neste sistema:

Administrador Responsável por criar, modificar e excluir promotores no sistema, possuindo acesso total à todos os eventos criados.

Promotor O promotor poderá criar, modificar e excluir eventos que pertençam à ele, à qualquer momento.

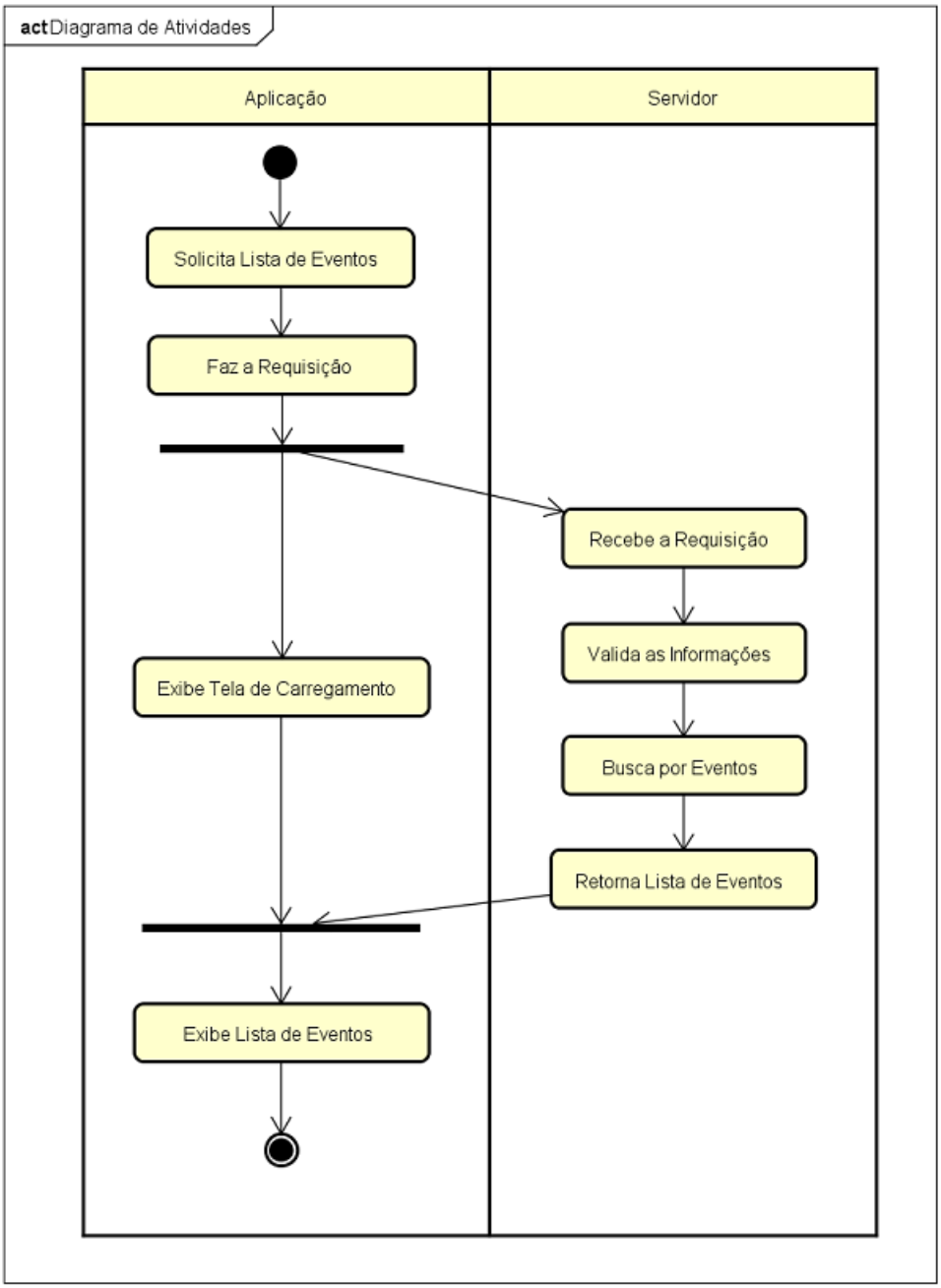
Usuário O usuário é aquele que instalou o aplicativo em seu aparelho, ele poderá registrar-se no sistema e fazer requisições de eventos próximos à ele ou de uma cidade inteira.

4.3.2 Diagrama de Atividade

Este diagrama de atividade específico mostra como funciona o principal caso de uso deste trabalho, que é a busca por eventos realizada pelo aplicativo em conjunto ao servidor de dados, representado pelo caso de uso “Obter Eventos”.

O aplicativo prepara a requisição e aguarda na tela de carregamento até que o servidor de dados responda a requisição para exibir a lista de eventos a tela. O diagrama de atividade está apresentado na Figura 20.

Figura 20 – Diagrama de Atividade do Caso de Uso “Obter Eventos”



Fonte: Elaborada pelo autor.

5 ESTUDO DE CASO

Para verificar a corretude do código e o funcionamento pleno do aplicativo e do site, neste capítulo será apresentado os resultados obtidos por meio de três estudos de caso realizados.

Estes estudos foram realizados na cidade de Dourados-MS, onde foram simulados eventos na cidade de Dourados - MS, Itaporã - MS e Nova Andradina - MS, com diferentes datas e localizações geográficas.

5.1 Preparação do ambiente de testes

Para fazer a função do servidor, foi utilizado um computador com processador 2x1.70GHz, 2 GigaBytes de memória RAM e 9.5 GigaBytes de tamanho total. O servidor foi executado sob a plataforma Linux, com sistema operacional Linux Mint 17.2 Cinnamon 64-bit. Neste computador, o servidor que responde as requisições do aplicativo e do site é uma instância de execução *Django Development*, conectado à um banco de dados PostgreSQL.

Para executar o aplicativo, foi utilizado um aparelho celular com processador 2x1.2 GHz, 2 GigaBytes de memória RAM, 8 GigaBytes de tamanho total, com uma tela de 5.0 polegadas, com um dispositivo GPS integrado e com uma placa de rede sem fio (Wi-Fi).

Após a definição dos dispositivos que fizeram o papel de servidor e usuário, foi criado uma série de dados de eventos e de promotores para a aplicação deste estudo de caso. Na Tabela 2, temos a especificação dos promotores que estão presentes neste banco de dados.

Tabela 2 – Tabela de promotores aplicado aos casos

Identificador	Nome do Promotor
1	João de Castro
2	Maria da Silva

Fonte: Elaborada pelo autor.

Para os eventos, na Tabela 3 temos a lista de eventos que foram utilizados nos testes. Os valores de localização (latitude e longitude) são valores reais e estão próximos de suas cidades, isto é indispensável para verificar a corretude da busca por GPS. Os dados de estilo do evento, site, banner e descrição foram ignorados por não influenciarem nos resultados do estudo de caso.

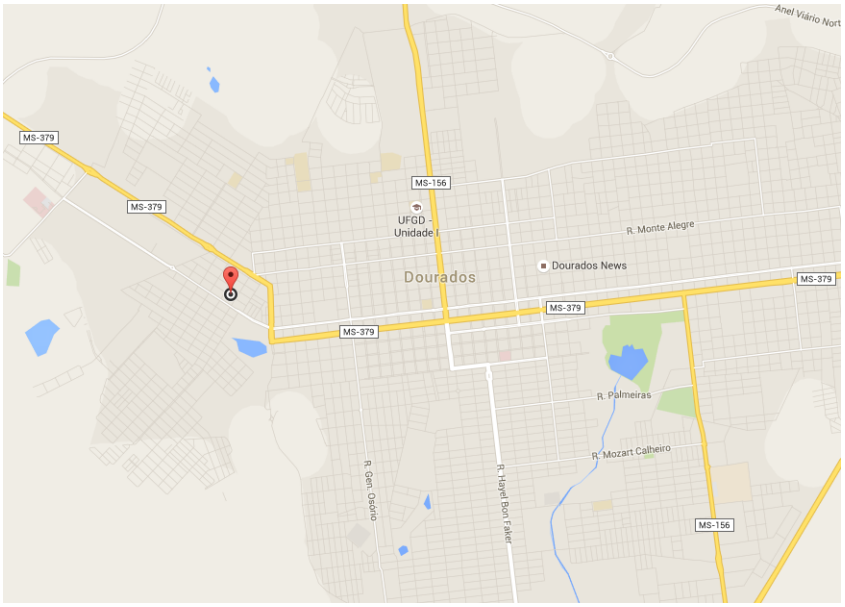
Tabela 3 – Tabela de eventos aplicado aos casos

Id	Id_Promotor	Título	Cidade	Localização	Data
1	1	Evento Teste 1	Dourados - MS	(-22.211912, -54.850023)	24/09/2015
2	1	Evento Teste 2	Itaporã - MS	(-22.081024, -54.792904)	24/09/2015
3	2	Evento Teste 3	Nova Andradina - MS	(-22.229107, -53.336944)	24/09/2015

Fonte: Elaborada pelo autor.

Durante a execução dos testes, a data e hora configurada no dispositivo era 24/09/2015, aproximadamente às 15:00. O dispositivo já possuía o aplicativo instalado, estava com uma conexão com a internet via rede sem fio (Wi-Fi) e estava exatamente na posição geográfica (-22.224807, -54.835074), na cidade de Dourados-MS. A Figura 21 mostra exatamente a posição geográfica em um mapa da cidade de Dourados - MS.

Figura 21 – Posição geográfica do aparelho de testes



Fonte: (GOOGLE, 2015), com alterações.

5.2 Execução dos testes

5.2.1 Caso 1

O primeiro caso testado foi utilizando um ator como o promotor de eventos que deseja inserir um evento no sistema. Para isso, o promotor de eventos segue os seguintes passos descritos na lista abaixo e apresentado na Figura 22:

1. Entra no site de gerenciamento de eventos;
2. Clica no link “Criar Evento”;

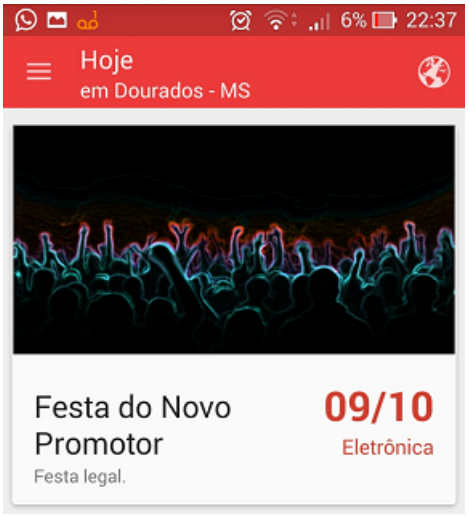
3. Preenche todos os dados necessários para o evento e clica em “Enviar”.

Figura 22 – Tela de preenchimento dos dados do evento

Fonte: Elaborada pelo autor.

De acordo com a Figura 23, os resultados dos passos no caso 1 trouxe o evento com o título **Festa do Novo Promotor**, que foi exatamente o que está especificado em seu registro.

Figura 23 – Resultado obtido no aplicativo com o caso 1



Fonte: Elaborada pelo autor.

5.2.2 Caso 2

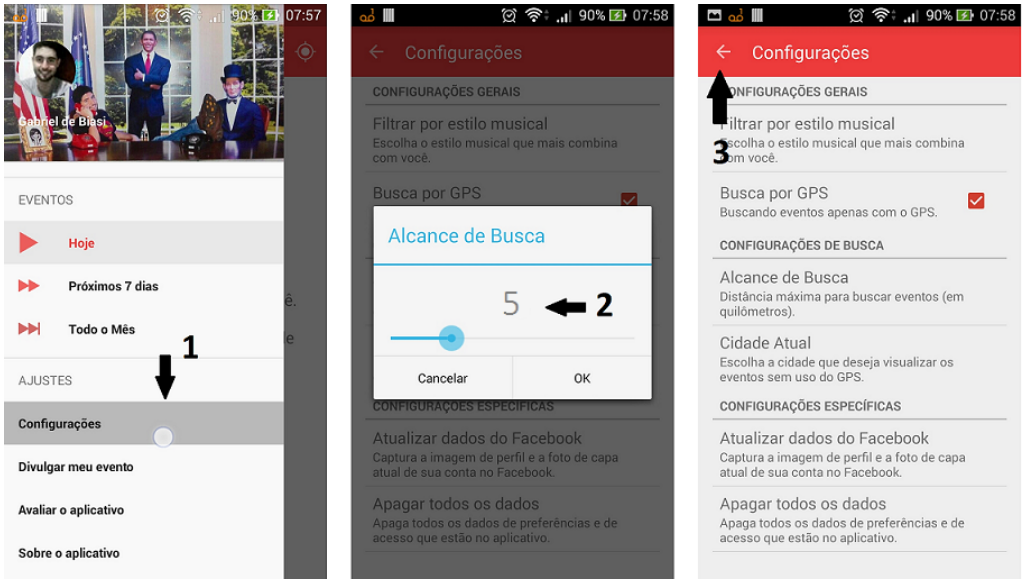
Neste segundo caso, o ator é um usuário que realiza a busca por eventos utilizando o GPS do dispositivo, fazendo com que apareça eventos que estejam à uma distância

máxima de 5 quilômetros.

Para isso, o usuário segue os seguintes passos descritos na lista abaixo e indicado na Figura 24:

1. Abre o menu lateral e toca na opção “Configurações”;
2. Escolhe o alcance de busca de 5 quilômetros;
3. Após isso, o usuário toca no botão voltar e a tela é atualizada.

Figura 24 – Passos da configuração do aplicativo para buscas com o GPS



Fonte: Elaborada pelo autor.

De acordo com a Figura 25, os resultados dos passos no caso 2 trouxe o evento com o título **Evento Teste 1**, que está de acordo com as regras da busca: A festa acontecerá em menos de 24 horas e está à menos de 5 quilômetros de distância do usuário.

Figura 25 – Resultado obtido no aplicativo com o caso 2

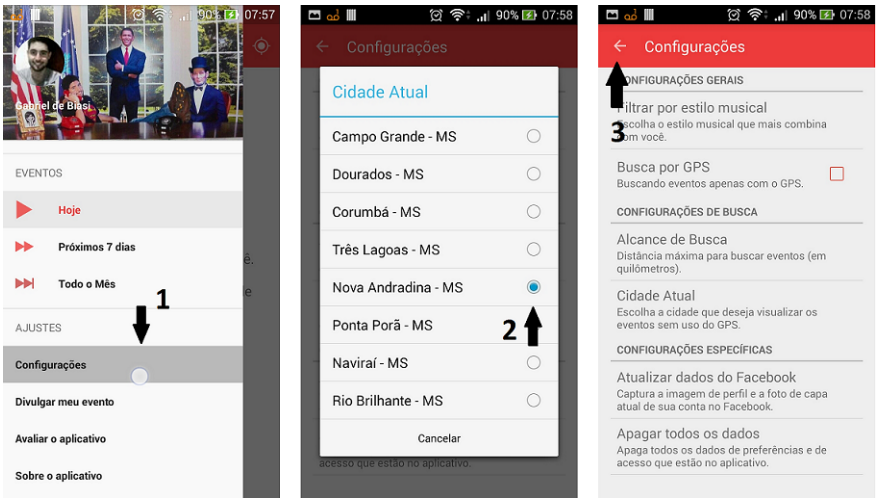
Fonte: Elaborada pelo autor.

5.2.3 Caso 3

No terceiro caso, o ator é um usuário que realiza a busca por eventos de uma cidade selecionada, neste exemplo, sendo a cidade de "Nova Andradina - MS". Para isso, o usuário segue os seguintes passos descritos na lista abaixo e indicado na Figura 26:

1. Abre o menu lateral e toca na opção "Configurações";
2. Seleciona a opção de busca por Cidade e então escolhe a cidade desejada;
3. Após isso, o usuário toca no botão voltar e a tela é atualizada.

Figura 26 – Passos da configuração do aplicativo para buscas em uma cidade



Fonte: Elaborada pelo autor.

De acordo com a Figura 27, os resultados dos passos no caso 3 trouxe o evento com o título **Evento Teste 3**, que está de acordo com as regras da busca: A festa acontecerá em menos de 24 horas e está localizada na cidade selecionada pelo usuário.

Figura 27 – Resultado obtido no aplicativo com o caso 3

Fonte: Elaborada pelo autor.

5.3 Análise dos testes

De acordo com os testes realizados, o aplicativo e o site mostraram os resultados esperados, fornecendo as informações detalhadas do evento pelo aplicativo ou inserindo com sucesso o evento no sistema através do site.

Os dados utilizados nestes testes foram simbólicos (exceto a localização geográfica), entretanto não há diferenças entre os dados utilizados para os possíveis dados verdadeiros de um evento real.

Durante os testes, o servidor de dados e o dispositivo móvel estavam na mesma rede local, para maior rapidez das respostas do teste. Porém, não há necessidade de adaptação de código no servidor e nem mesmo no aplicativo para que possam realizar as mesmas funções utilizando uma conexão com a internet.

No próximo capítulo, é feita a discussão final entre os objetivos propostos no início deste trabalho junto aos resultados obtidos neste estudo de caso.

6 CONCLUSÃO

Neste capítulo temos a discussão final sobre o trabalho realizado. Após a execução do estudo de caso, foi concluído que todos os objetivos deste trabalho foram executados com sucesso.

Entre os objetivos específicos concluídos estão o estudo do sistema operacional Android, que a partir dos referenciais teóricos foi possível desenvolver a aplicação totalmente funcional, o estudo da linguagem Python que proporcionou o desenvolvimento do servidor de dados do trabalho, a criação da documentação usando os conceitos da UML e a execução completa dos estudos de caso.

A implentação da aplicação móvel foi bem-sucedida, junto ao site de gerenciamento. O servidor de dados em sua versão atual responde as requisições da aplicação corretamente em um tempo hábil.

Para fins de comparação entre os métodos de divulgação discutidos, o uso desta aplicação móvel mostra-se eficiente em relação aos métodos tradicionais, por apresentar exatamente os eventos que o usuário deseja e que estão próximos geograficamente dele.

Estes motivos entram em contraponto aos métodos tradicionais, que podem divulgar eventos que estão à distâncias não informadas e podem atingir usuários que não são o público-alvo do evento.

6.1 Aplicação móvel

A aplicação móvel foi implementada sem muitas dificuldades, pelo fato da utilização do Android Studio para o desenvolvimento. Esta IDE facilitou a criação de classes e métodos, automatizando a codificação e correção de erros.

Em sua compilação final, a aplicação móvel precisa das seguintes permissões do aparelho para que possa funcionar corretamente: identidade do usuário, local preciso e aproximado do dispositivo, utilizar armazenamento externo (para *cache* de imagens dos eventos) e conexão com a internet.

6.2 Site de gerenciamento de eventos

A implementação do site de gerencimamento foi concluída, permitindo a realização do estudo de caso para testar as funcionalidades.

Como o site foi implementado utilizando o *framework* Django, código-fonte do site

de gerenciamento e do servidor de dados estão preparados para serem implantados em qualquer empresa que ofereça uma plataforma de suporte à Django junto ao banco de dados PostgreSQL.

6.3 Melhorias futuras

Nesta seção são apresentadas algumas melhorias que podem ser aplicadas à este trabalho futuramente.

Venda de ingresso Uma melhoria bastante interessante seria a possibilidade de oferecer a venda dos ingressos dos eventos divulgados diretamente pelo aplicativo. Isto significa uma economia de tempo para o usuário e uma nova proposta de modelo de negócio entre o administrador do sistema e o promotor de eventos.

Divulgação direta Utilizando as configurações de preferências dos usuários do aplicativo, a disseminação direta da informação de eventos, não necessariamente próximos ao usuário, poderia ser aplicado. Por exemplo, um usuário recebe uma notificação de um evento de seu estilo musical, mesmo se o evento não estiver no alcance estipulado pelo usuário.

Análise de dados Uma característica importante seria uma maneira de informar aos promotores de eventos como está a situação dos seus eventos no aplicativo. Por exemplo, quantidade de vezes que o evento foi visualizado ou quantidade de pessoas que possuem o aplicativo e que estão próximos ao evento perto do momento de início.

Cadastro de Promotores Uma maneira de facilitar a entrada de novos promotores seria permitindo o cadastro à partir do site de gerenciamento. Entretanto, seria necessário avaliar quais os critérios que seriam utilizados para permitir que uma pessoa pudesse se tornar um promotor de eventos.

Implementação para iOS e Windows Phone O Android não é o único sistema operacional para dispositivos móveis, portanto implementar o aplicativo para os sistemas iOS e Windows Phone pode aumentar a quantidade de usuários do sistema.

REFERÊNCIAS

ANDERSON, C. *Mapas com GeoDjango e PostGIS*. 2015. Disponível em: <http://christiano.me/mapas-com-geodjango-e-postgis/>. Acesso em: 11/08/2015.

ANDROID DEVELOPERS. *Dashboards | Android Developers*. 2015. Disponível em: <https://developer.android.com/about/dashboards/index.html>. Acesso em: 31/05/2015.

ANDROID DEVELOPERS. *Making Your App Location-Aware | Android Developers*. 2015. Disponível em: <https://developer.android.com/training/location/index.html>. Acesso em: 31/05/2015.

ANDROID DEVELOPERS. *Starting an Activity | Android Developers*. 2015. Disponível em: <http://developer.android.com/training/basics/activity-lifecycle/starting.html>. Acesso em: 31/05/2015.

ARMAL, S. *Sandip Armal's Blog: Architecture of Android*. 2015. Disponível em: <http://sanarmal.blogspot.com.br/2012/03/architecture-of-android.html>. Acesso em: 31/05/2015.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: Guia do Usuário*. [S.l.]: Elsevier, 2006.

DJANGO. *GeoDjango | Django documentation | Django*. 2015. Disponível em: <https://docs.djangoproject.com/en/1.8/ref/contrib/gis/>. Acesso em: 15/06/2015.

DJANGO. *The Web framework for perfectionists with deadlines | Django*. 2015. Disponível em: <https://www.djangoproject.com/>. Acesso em: 31/05/2015.

FACEBOOK. *Facebook Login for Android*. 2015. Disponível em: <https://developers.facebook.com/docs/facebook-login/android/v2.3>. Acesso em: 31/05/2015.

FACEBOOK. *Facebook SDK for Android*. 2015. Disponível em: <https://developers.facebook.com/docs/android>. Acesso em: 10/05/2015.

FIORINI, E. M. *Android Layouts: Aprendendo técnicas de Layout no Android*. 2015. Disponível em: <http://www.devmedia.com.br/android-layouts-aprendendo-tecnicas-de-layout-no-android/30790>. Acesso em: 31/05/2015.

GOOGLE. *Google Maps*. 2015. Disponível em: <http://maps.google.com.br/>. Acesso em: 01/09/2015.

HOLOVATY, A.; KAPLAN-MOSS, J. *The django book*. [S.l.: s.n.], 2007.

JSON. *Introducing JSON*. 2015. Disponível em: <http://json.org>. Acesso em: 10/08/2015.

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

LUTZ, M.; ASCHER, D. *Aprendendo Python, 2.* [S.l.]: Bookman, 2007.

PYTHON SOFTWARE FOUNDATION. *About Python | Python.org.* 2015. Disponível em: <https://www.python.org/about/>. Acesso em: 31/05/2015.

SOURCEFORGE. *Uma breve história do PostgreSQL.* 2015. Disponível em: <http://pgdocptbr.sourceforge.net/pg80/history.html>. Acesso em: 10/10/2015.

SROKA, R. *Model-View-Controller design pattern in iOS development.* 2015. Disponível em: <http://stunningco.de/2011/12/25/model-view-controller-design-pattern-in-ios-development/>. Acesso em: 31/05/2015.

Apêndices

APÊNDICE A – CÓDIGO-FONTE DO MÉTODO APP_REQUEST

Neste apêndice está um trecho importante do código-fonte na linguagem Python que trata requisição de eventos feita pelo aplicativo no servidor de dados. O servidor Django realiza a chamada do método *app_request(request)* para responder todas as requisições de qualquer usuário.

```
1 # -*- coding: utf-8 -*-
2 def app_request(request):
3   # Apenas requisicoes POST sao aceitas.
4   if request.method == 'POST':
5
6     # Verifica se o usuario existe no servidor.
7     user_id = request.POST.get('facebook_id', '-1')
8     user = get_object_or_404(Profile, facebook=user_id)
9
10    # Incrementa o contador de requisicoes deste usuario.
11    user.reqs += 1
12
13    # Uso de filtro por estilo do evento.
14    style = int(request.POST.get('style', '-1'))
15    if style > 0:
16      style_Q = Q(style=style)
17    elif style == 0:
18      style_Q = Q()
19    else:
20      return HttpResponseBadRequest()
21
22    # Processa o intervalo de datas
23    mode = int(request.POST.get('viewMode', '-1'))
24    now = timezone.now()
25    # HOJE
26    # Mostra todos os eventos ate as 23:59:59 do dia atual.
27    if mode == 0:
28      futuro = datetime.combine(now+timedelta(days=1), time(0,0)
          )
```

```
29          data_Q = Q(data__range=[now, futuro])
30
31      # PROXIMOS DIAS
32      # Mostra os eventos para 7 dias.
33      elif mode == 1:
34          data_Q = Q(data__range=[now, now+timedelta(days=7)])
35
36      # MES INTEIRO
37      # Mostra eventos do mes.
38      elif mode == 2:
39          data_Q = Q(data__range=[now, now+timedelta(days=30)])
40
41      # MODO CHEIO (sem filtro de data)
42      # Implementado para testes.
43      elif mode == 3:
44          data_Q = Q()
45
46      # SEM MODO
47      # Retorna BadRequest
48      else:
49          return HttpResponseBadRequest()
50
51      cidade = request.POST.get('cidade', '-1')
52      # Se a cidade nao for especificada, utiliza-se os dados de
            GPS.
53      if cidade == '-1':
54          # BUSCA POR GPS
55          latitude = float(request.POST.get('latitude', '0'))
56          longitude = float(request.POST.get('longitude', '0'))
57          distance = int(request.POST.get('distance', '0'))
58          point = Point(longitude, latitude)
59          user.location = point
60          location_Q = Q(location__distance_lte=(point, D(km=
                distance)))
61      else:
62          # BUSCA POR CIDADE
63          user.cidade = cidade
64          location_Q = Q(cidade=cidade)
65
```

```
66      resultado = Festa.objects.filter(data_Q & style_Q &
            location_Q)
67      user.save()
68      # Resultados sao retornados em formato JSON.
69      return HttpResponse(serializers.serialize("json", resultado)
            , content_type='application/json')
70
71  # Se algo der errado, retorna BadRequest.
72  return HttpResponseBadRequest()
```