UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

RAPHAEL JARDIM LOPES

SISTEMA DE COMPARTILHAMENTO DE IMAGENS GEOLOCALIZADAS PARA DISPOSITIVOS MÓVEIS

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

2015

RAPHAEL JARDIM LOPES

SISTEMA DE COMPARTILHAMENTO DE IMAGENS GEOLOCALIZADAS PARA DISPOSITIVOS MÓVEIS

Trabalho de Conclusão de Curso de Sistemas de Informação apresentado ao Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná como requisito para obtenção do título de "Bacharel em Sistemas de Informação".

Orientadora: Prof^a. Dr^a. Ana Cristina Barreiras Kochem Vendramin

CURITIBA

2015

A minha mãe, Maria Dolores, que estará sempre no meu coração.

AGRADECIMENTOS

Agradeço a minha orientadora Prof^a. Dr^a. Ana Cristina Barreiras Kochem Vendramin, pelo apoio que me deu no desenvolvimento deste Trabalho de Conclusão de Curso.

Agradeço também a minha esposa, Helena Oliveira, que me incentivou e deu apoio nessa jornada.

RESUMO

LOPES, Raphael Jardim. SISTEMA DE COMPARTILHAMENTO DE IMAGENS GEOLOCALIZADAS PARA DISPOSITIVOS MÓVEIS. 74 f. Trabalho de Conclusão de Curso – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

Neste trabalho é proposto o desenvolvimento de um sistema para compartilhamento de imagens geolocalizadas por meio de dispositivos móveis. O enfoque principal do sistema é o compartilhamento de fotos de paisagens e pontos turísticos, incentivando a descoberta e visita a lugares interessantes de diversas regiões do mundo por qualquer pessoa que utilize o sistema. A popularização da internet e a evolução na área dos dispositivos móveis, permitiu a popularização dos *smartphones* e do compartilhamento de imagens. Ainda, o advento das redes de telefonia móvel e sem fio, permite que as pessoas possam ter acesso a informações quase que instantaneamente, independente de onde estiverem. Dentre os recursos disponíveis para as pessoas nos seus *smartphones*, estão principalmente a troca de informações entre elas, facilitando assim o chamado *crowdsourcing*. O *crowdsourcing* ocorre quando o preenchimento das informações de um sistema é feito de forma comunitária, por parte das pessoas que utilizam o sistema. Outro recurso fornecido pelos *smartphones* é a capacidade de tirar fotos, incluindo também dados de localização nas mesmas. Este trabalho utiliza os recursos de geolocalização e redes sem fio dos aparelhos celulares para criar o sistema proposto.

Palavras-chave: fonte de informação de multidão, geolocalização, fotos, celular inteligente

ABSTRACT

LOPES, Raphael Jardim. GEOLOCATION IMAGE-SHARING SYSTEM FOR MOBILE DEVICES. 74 f. Trabalho de Conclusão de Curso – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

In this work it is proposed the development of a geolocation-based image sharing system for mobile devices. The main focus is the sharing of landscapes' photos and tourist hotspots, encouraging the discovery and visitation to interesting places around the world. The popularization of the internet and the evolution in the field of mobile devices, has allowed for the popularization of the smartphones with image sharing features. Furthermore, the advent of mobile and wireless technology has given people access to information almost instantly, independent from where they are. With the prevalence of smartphones bringing the ease of information sharing between people, we have seen the rise of the phenomenon known as crowdsourcing. Crowdsourcing occurs when the supply of information of a system is achieved communally. Another resource provided by smartphones is the capacity of taking pictures with geolocation data. This work employs the geolocation and wireless network resources available in the smartphones to create the proposed system.

Keywords: crowdsourcing, geolocation, photos, smartphones

LISTA DE FIGURAS

FIGURA 1 – Tela inicial do iOS 7. 18
FIGURA 2 – Camadas do iOS. 18
FIGURA 3 – Interface do Panorâmio 21
FIGURA 4 – Mapa de Imagens do Flickr 22
FIGURA 5 – Mapa de Imagens do Instagram 23
FIGURA 6 – Exemplo de sintaxe do Tour of Go 28
FIGURA 7 – Xcode Storyboard. 32
FIGURA 8 – Editor de Texto Atom. 33
FIGURA 9 – Mapeamento da superfície da terra para uma superfície plana. 38
FIGURA 10 – Tela Inicial 50
FIGURA 11 – Tela do Mapa. 51
FIGURA 12 – Tela do Mapa com imagem selecionada. 52
FIGURA 13 – Tela de detalhes da foto. 53
FIGURA 14 – Tela de busca de fotos. 54
FIGURA 15 – Tela de fotos. 55
FIGURA 16 – Tela de fotos com imagem selecionada. 56
FIGURA 17 – Tela do formulário de envio de foto. 57
FIGURA 18 – Tela exibida enquanto a foto está sendo enviada. 58
FIGURA 19 – Tela do perfil do usuário. 59
FIGURA 20 – Tela de edição do perfil do usuário. 60
FIGURA 21 – Diagrama de Casos de Uso. 69

LISTA DE TABELAS

TABELA 1 – Recursos de hardware 34
TABELA 2 – Recursos de software 35

LISTA DE SIGLAS

GPS	*Global Position System*
U.M.L.	*Unified Modeling Language*
SDK	*Software Development Kit*
IBM	*International Business Machine*
PC	*Personal Computer*
HTTP	*Hypertext Transfer Protocol*
WWDC	WorldWide Developers Conference
SGBD	Sistema de Gerenciamento de Banco de Dados
WSDL	Web Services Description Language
XML	*Extensible Markup Language*
SQL	Structured Query Language
SOAP	*Simple Object Access Protocol*
REST	Representational State Transfer
JSON	JavaScript Object Notation
URL	Uniform Resource Location

SUMÁRIO

1 INTRODUÇÃO 12
1.1 MOTIVAÇÃO 13
1.2 OBJETIVO GERAL E OBJETIVOS ESPECÍFICOS 13
1.3 ESTRUTURA E ORGANIZAÇÃO DO DOCUMENTO 13
2 LEVANTAMENTO BIBLIOGRÁFICO 15
2.1 CROWDSOURCE 15
2.2 FOLKSONOMIA 15
2.3 SMARTPHONE 16
2.3.1 iOS 16
2.4 GEOPROCESSAMENTO 19
2.4.1 Trabalhos relacionados 20
2.4.1.1 Google Panoramio 21
2.4.1.2 Flickr 21
2.4.1.3 Instagram 22
2.4.1.4 Picasa 23
2.4.1.5 Discussões 24
3 METODOLOGIA 25
3.1 TECNOLOGIAS 25
3.1.1 iOS SDK 26
3.1.2 Linux 26
3.1.3 Linguagem de Programação Go 27
3.1.4 Swift 28
3.1.5 Banco de dados 29
3.1.5.1 PostgreSQL 30
3.1.6 Web Services 30
3.1.7 XCode 31
3.1.8 Atom 32
4 RECURSOS DE HARDWARE E SOFTWARE 34
4.1 RECURSOS DE HARDWARE 34
4.2 RECURSOS DE SOFTWARE 35
5 DESENVOLVIMENTO DO SISTEMA 36
5.1 IOS SDK 36
5.1.1 Framework de Mapas 37
5.1.1.1 Convertendo entre sistemas de coordenadas 39
5.1.2 Core Location 40
5.1.2.1 Iniciando o serviço de localização padrão 41
5.2 COMUNICAÇÃO ENTRE APLICATIVO MÓVEL E SERVIDOR 43
5.3 SOFTWARE SERVIDOR 44
5.3.1 Recebendo fotos dos clientes 44
5.3.2 Buscando fotos com base na localização 46
5.4 INTERFACE GRÁFICA 49
5.4.1 Tela Inicial 49

5.4.2 Tela do Mapa 50
5.4.3 Tela de Busca de fotos 53
5.4.4 Tela de Fotos 54
5.4.4.1 Tela de Envio de Foto 56
5.4.5 Tela de Perfil do usuário 58
6 CONCLUSÃO 61
6.1 TRABALHOS FUTUROS 62
REFERÊNCIAS 64
Apêndice A – PROJETO DE SOFTWARE 67
A.1 LEVANTAMENTO DE REQUISITOS 67
A.1.1 Requisitos funcionais 67
A.1.2 Requisitos não funcionais 68
A.2 CASOS DE USO 69
A.2.1 Diagrama de casos de uso 69
A.2.2 UC1 - Cadastro de Usuários 70
A.2.3 UC2 - Autenticação de Usuário 70
A.2.4 UC3 - Adição de imagem 71
A.2.5 UC4 - Adição de palavras-chave 71
A.2.6 UC5 - Adição de nota 72
A.2.7 UC6 - Visualizar imagens 72
A.2.8 UC7 - Busca de imagens 73
A.2.9 UC8 - Denúncia de imagens 73

1 INTRODUÇÃO

O compartilhamento de imagens nunca foi tão fácil, principalmente por usuários de redes sociais que tiram fotos de praticamente todos os locais e momentos, utilizando *smartphones* para compartilhar essas imagens através da internet (HOCHMAN; SCHWARTZ, 2012).

Tendo em vista que a maioria das fotos tiradas por *smartphones* já estão geolocalizadas através de informações contidas nas próprias imagens que tem dados EXIF[1] embutidos (CAMERA; ASSOCIATION, 2012), várias comunidades de compartilhamento de fotos já fazem uso destas informações para melhorar o modo em que disponibilizam seu conteúdo.

Existem diversos aplicativos na área de turismo que utilizam dados EXIF e estes estão disponíveis nas principais lojas de aplicativos móveis (APPLE, 2014a)(GOOGLE, 2014a). Porém, os aplicativos móveis focam principalmente na compra de passagens, hospedagem e GPS (*Global Position System*), como é o caso dos aplicativos que aparecem na lista de mais populares em outubro de 2014 das lojas da Apple and Google, dentre eles: Google Earth, Waze Social GPS e Hotel Urbano.

Com vista no que foi exposto anteriormente, o presente trabalho tem por objetivo desenvolver um serviço que ajude as pessoas a compartilhar fotos e informações que achem úteis de locais em suas cidades e em cidades que estejam visitando, dando a sua perspectiva única de cada local. O trabalho fará uso de *smartphones* como centro dessa troca de informações, pois, a maioria deles vem com mapas e dispositivos de localização embutidos, facilitando assim a localização de fotos e outras informações nas redondezas, como por exemplo, um ponto turístico desconhecido ou não muito conhecido.

[1] *Exchangeable Image File Format* é um padrão que especifica formatos para imagens, sons e metadados seguida por fabricantes de câmeras digitais (incluindo smartphones), scanners e outros dispositivos. É especificado que informações serão gravadas junto ao arquivo propriamente dito, um exemplo do que é especificado são os metadados de localização geográfica (CAMERA; ASSOCIATION, 2012).

1.1 MOTIVAÇÃO

A principal motivação para o desenvolvimento deste projeto é implementar um aplicativo de compartilhamento de fotos em dispositivos móveis - *smartphones* que facilite a troca de informações entre as pessoas de locais interessantes em suas cidades e em cidades que estejam visitando. Nesse aplicativo, os próprios usuários preenchem o conteúdo do serviço, para benefício mutuo. Para isso, o projeto utiliza-se de tecnologias de geolocalização, mapas e a capacidade dos dispositivos móveis de tirar fotos, além da capacidade de processamento e armazenamento de informações dos computadores atuais.

1.2 OBJETIVO GERAL E OBJETIVOS ESPECÍFICOS

O objetivo geral deste trabalho é criar um software que utiliza tecnologia móvel para facilitar o descobrimento de pontos turísticos. Para atender esse objetivo, serão utilizadas imagens geolocalizadas de regiões, locais e paisagens compartilhadas pelos usuários do próprio aplicativo. Os objetivos específicos do presente trabalho são:

- Estudar tecnologias de desenvolvimento para dispositivos móveis e de *backend*;
- Desenvolver um sistema com arquitetura cliente/servidor que permita o armazenamento e o processamento geográfico de imagens compartilhadas por usuários;
- Desenvolver um sistema que permita registrar imagens e marcar as localizações das imagens com palavras-chave para facilitar a busca por tais imagens;
- Desenvolver um sistema que permita buscar imagens por regiões ou através de palavras-chave;
- Popular a base de dados com imagens geolocalizadas de uma cidade;
- Desenvolver um protótipo do sistema para dispositivos móveis, para que as pessoas possam utilizá-lo para buscar e compartilhar imagens de pontos turísticos;

1.3 ESTRUTURA E ORGANIZAÇÃO DO DOCUMENTO

O presente trabalho está dividido em cinco capítulos. O presente capítulo apresenta brevemente os objetivos e a motivação do desenvolvimento. O segundo capítulo apresenta a base teórica do trabalho incluindo os principais conceitos e o estado da arte. O terceiro capítulo apresenta a metodologia utilizada para o desenvolvimento do trabalho.

O quarto capítulo descreve os recursos e as ferramentas de software e hardware utilizados no desenvolvimento do presente trabalho, bem como os mesmos foram adquiridos. O quinto capítulo apresenta itens pertinentes ao desenvolvimento do protótipo, tais como as APIs utilizadas. As conclusões são apresentadas no sexto capítulo. Por fim, no apêndice se encontra o projeto de software, incluindo os requisitos funcionais e não funcionais, e a descrição dos casos de uso.

2 LEVANTAMENTO BIBLIOGRÁFICO

Este capítulo apresenta informações resultantes do levantamento bibliográfico realizado para o desenvolvimento deste trabalho. São apresentados alguns conceitos e informações que são utilizados para dar uma melhor visão do projeto.

2.1 CROWDSOURCE

Segundo Chaianuchittrakul (2013), *crowdsource* é uma abordagem para lidar com problemas complexos ou atingir metas com uma grande rede de participantes (*crowd*).

Howe e Robinson (2006) dizem que "*crowdsourcing* representa o ato de uma empresa ou instituição tirar uma função antes realizada por seus funcionários e a terceirizar para uma rede indefinida (e geralmente grande) de pessoas na forma de uma chamada para contribuições". Em *crowdsourcing*, a tarefa principal é geralmente quebrada em várias tarefas menores chamada microtarefas. Estes tipos de microtarefas permitem que os trabalhadores as realizem de maneira mais rápida, barata e com menos erros. *Crowdsourcing* pode ser realizado através de tecnologias web o que permite aos indivíduos trabalharem em um mesmo ambiente com restrições descentralizadas como, por exemplo, pessoas de locais diferentes, com costumes e culturas diferentes (CHAIANUCHITTRAKUL, 2013).

2.2 FOLKSONOMIA

Folksonomia é uma forma de aproveitar o conhecimento de pessoas de uma maneira democrática para organizar e classificar os recursos que estão na internet, tornando-se uma construção colaborativa onde várias pessoas chegam a um acordo sobre um mesmo recurso (PIRAQUIVE et al., 2009). Folksonomia é mais formalmente conhecida como *collaborative tagging* (GOLDER; HUBERMAN, 2006).

Através da folksonomia, portanto, empregamos *tags* em um determinado recurso com o intuito de categorizá-lo. As *tags* são escolhidas somente pelos usuários sem nenhuma

intervenção da máquina. Apesar disso, observa-se alguns mecanismos como a rede social *LastFM*, que é um serviço de recomendação musical, utiliza de um sistema chamado Scrobbler funciona enviando as músicas geralmente ouvidas pelo usuário para construir um perfil detalhado do usuário, podendo assim recomendar músicas semelhantes ao do gosto do usuário (LASTFM, 2015). Ele recomenda *tags* no momento da categorização. As *tags* utilizadas durante a categorização serão úteis no processo de busca tanto pelo próprio usuário que categorizou tal recurso quanto para outros usuários (BARCZYSZYN; LOPES, 2013).

2.3 SMARTPHONE

Segundo Theoharidou et al. (2012), o termo *smartphone* é frequentemente utilizado pela indústria e pela comunidade de pesquisadores para se referir a aparelhos celulares considerados "inteligentes" e que são distinguidos de aparelhos normais que tem tecnologia restrita. Esses últimos, referidos como *feature phones*, são geralmente restringidos por telas pequenas, capacidade de processamento e de rede limitados, e executam, no geral, sistemas operacionais proprietários e não adequadamente documentados. Portanto, a segurança dos mesmos é baseada principalmente no sigilo ou como a comunidade de segurança da informação se refere, "segurança por obscuridade".

Uma definição alternativa de *smartphone* é:

> "*smartphone* é um aparelho celular com capacidades avançadas, que executam um sistema operacional identificável permitindo aos usuários expandirem suas funcionalidades com aplicações de terceiros que estão disponíveis em um repositório de aplicações". (THEOHARIDOU et al., 2012)

.

O iPhone foi o *smartphone* escolhido para o desenvolvimento desse projeto, pois o autor já tem familiaridade com o ecosistema da Apple, e também pelo fato de que atualmente os apps (aplicativos) faturam mais na *Apple Store* do que na *Play Store* dos sistemas Android (COMPANY, 2015). A seguir é apresentado o sistema operacional utilizado pelos *smartphones* da Apple.

2.3.1 IOS

Anteriormente conhecido como iPhone OS, o iOS é o sistema operacional móvel desenvolvido pela Apple inc. e distribuído exclusivamente para o *hardware* da Apple.

Tendo sua primeira versão apresentada em 2007 com o anúncio do iPhone pela Apple, o iOS compartilha com o OS X[1] alguns *frameworks* como *Core Foundation*[2] e *Foundation*[3], entretanto sua interface gráfica é chamada de *Cocoa Touch* ao contrário de *Cocoa* do OS X. Com isso, ele provê um *framework* de interface gráfica diferente do sistema operacional da Apple para *desktops* não podendo executar aplicações feitas para *deisktop*. Enquanto o iOS também compartilha sua fundação no kernel *Darwin*, utilizado em ambos OS X e iOS, no iOS o shell *Unix-like* não está disponível para usuários e é restrito para aplicações (apps) fazendo com que o mesmo não seja totalmente compatível com Unix.

A interface de usuário do iOS é baseada no conceito de manipulação direta, utilizando-se de gestos multitoque. Alguns dos elementos de controle da interface consistem de botões deslizáveis, *switches*, e botões normais. Interações com o Sistema operacional incluem gestos como toques, *swipe*[4], *pinch*[5] e *pinch* reverso[6], todos tendo definições específicas dentro do contexto do sistema operacional e sua interface multitoque. Acelerômetros internos são usados por algumas aplicações para responder a balanços no aparelho (um resultado comum é o comando de desfazer) ou rotacioná-lo em três dimensões (o resultado comum é a mudança do modo retrato para paisagem).

A Figura 1 exibe a tela inicial do iOS 7.

[1] OS X é o sistema operacional da Apple para *desktops e notebooks*, baseado no UNIX, vem instalado em todas as maquinas da Apple, sucessor do sistema operacional Mac OS 9.

[2] Core Foundation é um *framework* que provê abstrações para tipos comuns de dados, facilita a internacionalização através do padrão Unicode e fornece um conjunto de utilidades, como por exemplo um sistema de plugins, preferêncas.

[3] O *framework Foundation* define a camada base das classes da linguagem de programação Objective-C e Swift. Além de oferecer um conjunto de classes primitivas úteis, o mesmo introduz vários paradigmas que não são cobertos pela linguagem Objective-C ou Siwft. Podemos citar uma delas a classe básica NSObject, que define as funções isEquals, hash and self.

[4] *Swipe* é um gesto em que o usuário desliza o dedo pela tela para ativar alguma funcionalidade. Uma das utilizações desse gesto é a função de aumentar e diminuir o volume.

[5] *Pinch* é um gesto em que o usuário "belisca"a tela do *smartphone*. Uma das principais utilizações do gesto Pinch é dar zoom em uma imagem.

[6] *Pinch* reverso é o gesto em que o usuário coloca os dois dedos na tela e separa os dois em direções contrárias, como um "beliscão"ao contrário. Uma das principais utilizações deste gesto é diminuir o zoom de uma imagem.

Figura 1: Tela inicial do iOS 7.
Fonte: davinci1993 (2014)

A arquitetura do iOS é divida em camadas, como pode ser visto na Figura 2.

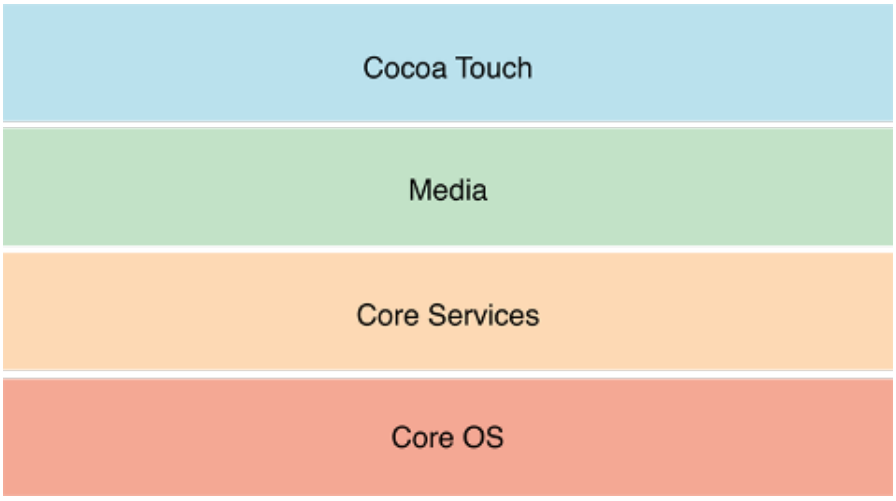

Figura 2: Camadas do iOS.
Fonte: Apple (2014b)

No nível mais alto tem-se a camada *Cocoa Touch* onde estão localizados os *frameworks* que definem a aparência das aplicações. É nesta camada que há o suporte a

tecnologias chave como multitarefas, notificações e vários outros serviços de alto nível.

Na camada *Media* tem-se as tecnologias de gráficos, áudio e vídeo que permitem a implementação de recursos multimídia nas aplicações iOS.

Na camada intermediária tem-se o *Core Services* que contém serviços fundamentais para os aplicativos. Entre esses serviços chave, pode-se citar os *frameworks Core Foundation* e *Foundation*, os quais definem tipos básicos que todas as aplicações utilizam. Podemos citar as seguintes classes que fazem parte do *Foundation* framework: NSString, NSDate, NSArray, que provem funcinalidades básicas, como armazenamento de informações, representam cadeias de caracteres e facilitam a manipulação de datas.

Por fim, temos a camada *Core OS*, que contém os recursos de baixo nível que a maioria das outras camadas constroem suas abstrações (APPLE, 2014b).

De um modo geral, pode-se dizer que no nível mais alto, o iOS age como um intermediário entre o hardware subjacente e os aplicativos. Aplicativos não podem conversar com o hardware subjacente diretamente. Em vez disso, eles se comunicam com o hardware através de um conjunto bem definido de interfaces do sistema. Estas interfaces tornam fácil a escrita de aplicações que trabalham consistentemente em aparelhos de diferentes capacidades.

As camadas de alto nível utilizam os serviços das camadas de mais baixo nível e promovem serviços e tecnologias mais sofisticadas, pois, as mesmas estão disponíveis para prover abstrações orientadas a objeto dos recursos de baixo nível. Estas abstrações geralmente tornam fácil construir aplicações, reduzindo a quantidade de código que é necessário para desenvolvê-las, encapsulando recursos potencialmente complexos, como *sockets* e *threads* (APPLE, 2014b).

2.4 GEOPROCESSAMENTO

Geoprocessamento é uma área que envolve a utilização de um conjunto de técnicas que permite fazer análises espaciais, manipular e gerenciar informações georreferenciadas (MOURA, 2003).

Segundo Câmara e Monteiro (2004) o termo Geoprocessamento denota a:

> "Disciplina do conhecimento que utiliza técnicas matemáticas e computacionais para o tratamento da informação geográfica e que vem influenciando de maneira crescente as áreas de Cartografia, Análise de Recursos Naturais, Transportes, Comunicações, Energia e Planejamento Urbano e Regional."

O dado geográfico é descrito sempre pela existência de uma característica nas entidades que a associa com uma localização geográfica existente ou com uma outra entidade geográfica cuja localização geográfica é conhecida. Esses dados podem ser coordenadas geográficas, endereços completos, referências únicas como o nome de um shopping, ou pode indicar o estado onde a entidade se encontra (LI; TORRES, 2014).

Os sistemas de informações geográficas (GIS) são ferramentas computacionais para o geoprocessamento que permitem realizar análises complexas, ao integrar diversas fontes de dados e criar bancos de dados geo-referenciados. Os GIS possibilitam ainda a automatização da produção de documentos cartográficos (CÂMARA; MONTEIRO, 2004).

Câmara e Monteiro (2004) também descreve um arcabouço conceitual para que se possa entender o processo de tradução do mundo real para um modelo computacional de geoprocessamento. Dividindo-o em quatro universos:

- o universo do *mundo real* que inclui as entidades da realidade a serem modeladas no sistema;
- o universo do *mundo matemático* (conceitual) que inclui uma definição matemática (formal) das entidades a serem representadas;
- o universo de *representação* onde as diversas entidades formais são mapeadas para representações geométricas e alfanuméricas no computador;
- o universo de *implementação* onde as estruturas de dados e algoritmos são escolhidos, baseados em considerações como desempenho, capacidade do equipamento e tamanho da massa de dados. É neste nível que acontece a codificação.

A visão dos universos apresentada não se limita apenas aos sistemas de geoprocessamento, podendo representar uma perspectiva unificadora para os problemas de Computação Gráfica e Processamento de Imagens (CÂMARA; MONTEIRO, 2004).

2.4.1 TRABALHOS RELACIONADOS

Nas seções a seguir são apresentados os trabalhos relacionados ao sistema proposto.

2.4.1.1 Google Panoramio

Panoramio é um serviço de compartilhamento de imagens geolocalizadas, adquirido pela Google em julho de 2007. Panoramio pede aos usuários para organizarem imagens usando *tags* (o mesmo utiliza folksonomia) que permitem a busca de imagens sobre algum tópico, como por exemplo nome do local ou assunto. Panoramio também foi uma das primeiros páginas da internet a implementar *cloud tags*[7] que provem acesso a imagens marcadas com as palavras-chave mais populares. O Panoramio também hospeda uma lista de locais famosos ao redor do mundo.

A Figura 3 exibe a página inicial do Panoramio, permite visualizar diretamente do navegador as fotos no mapa, selecionar as mais populares, ou explorar o mapa.

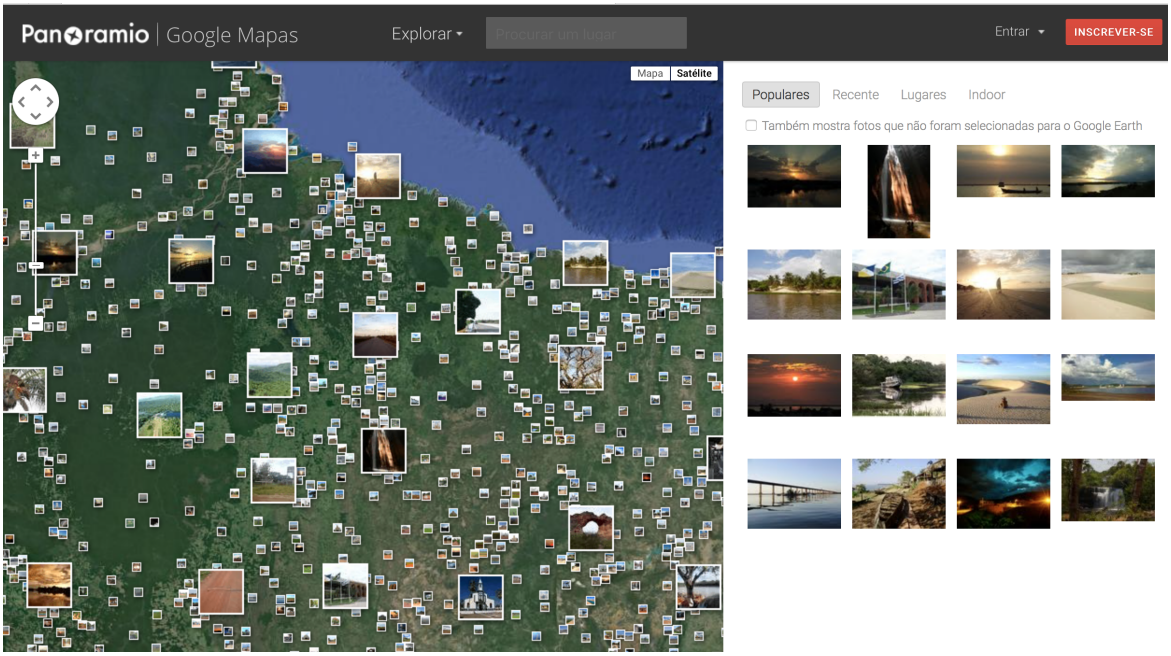

Figura 3: Interface do Panorâmio

2.4.1.2 Flickr

Flickr é um serviço de hospedagem de imagens e vídeos, fundado pela Ludicorp em 2004 e comprado pelo Yahoo em 2005 (FLICKR, 2015). No Flickr, fotos podem conter geotags e qualquer album que tenha alguma foto com geotag pode ser exibido em um mapa usando o serviço Imapflickr, que é um serviço que permite selecionar imagens do flickr e criar um mapa personalizado, que pode ser embutido em um site ou blog, pode também ser enviado para o Facebook e Twitter (DIGITAL, 2014). Há também a opção

[7] Uma representação visual de dados textuais, tipicamente utilizado para descrever metadados de palavras-chave em páginas da web. Tags são geralmente de uma única palavra e a importância de cada tag é mostrada através da cor ou do tamanho da fonte.

de visualizar imagens em um mapa fornecido pelo próprio serviço.

A Figura 4 exibe o mapa de imagens geolocalizadas no Flickr, nela é possível ver as imagens georreferenciadas, a quantidade total de itens, classificar por categorias e buscar no mapa.

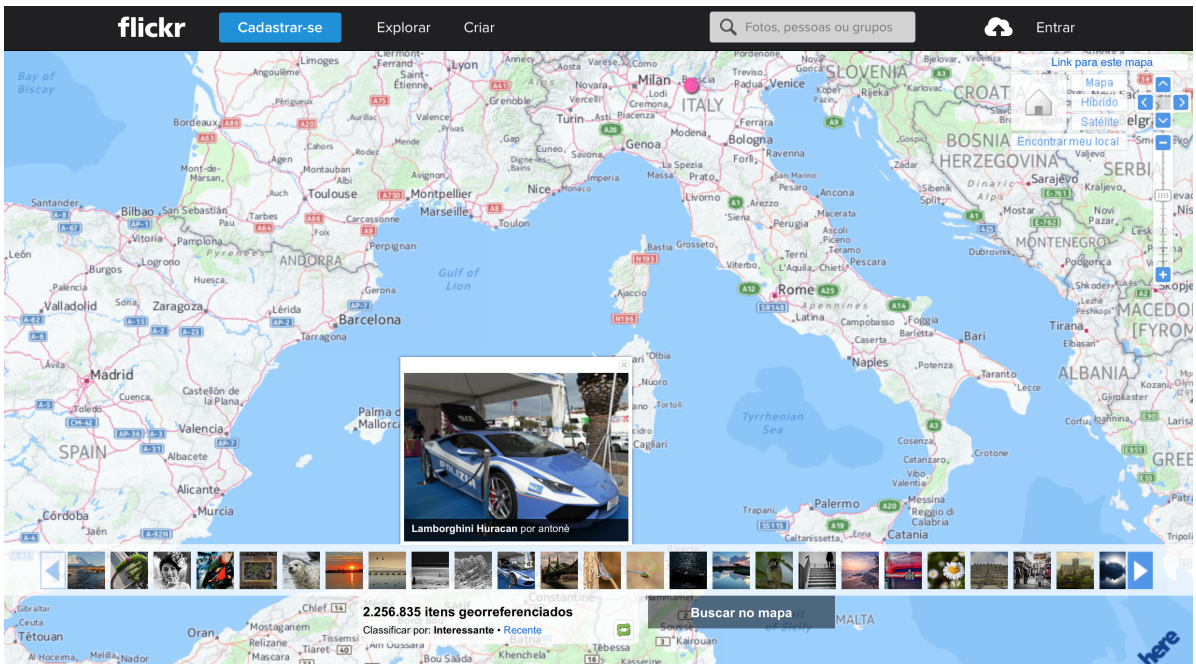

Figura 4: Mapa de Imagens do Flickr

Fonte: Flickr (2015)

2.4.1.3 Instagram

Instagram é uma rede social online de compartilhamento de foto e vídeo, criado por Kevin Systrom e Mike Krieger e lançada em outubro de 2010. Em 9 de abril de 2012, o Facebook adquiriu o Instagram. Também é possível exibir locais onde as fotos dos usuários foram tiradas. Entretanto, é necessário habilitar uma opção para que as fotos sejam exibidas no mapa. O Instagram utiliza as coordenadas geográficas do local em que a foto foi tirada quando esta informação está disponível ou o usuário pode inserir o local depois de ter feito o envio da foto (INSTAGRAM, 2014).

A Figura 5 exibe o mapa de imagens geolocalizadas no Instagram. Como pode ser visto, caso o usuário não tenha compartilhado nenhuma foto, não há nenhuma opção de buscar por outras fotos.

Figura 5: Mapa de Imagens do Instagram

2.4.1.4 Picasa

Picasa é um software que inclui a edição digital de fotografias e cuja função principal é organizar fotos digitais presentes no computador, de forma a facilitar a procura por fotografias específicas por parte do usuário. Foi criado pela empresa Picasa, Inc. e adquirida em julho de 2004 pelo Google. A partir de então, a empresa norte-americana passou a oferecer o programa gratuitamente em sua página na Internet. É possível visu-

alizar as fotos do Picasa no Google Earth através de geotags, exportando os dados para um arquivo do Google Earth (GOOGLE, 2014b).

2.4.1.5 Discussões

O presente capítulo apresentou alguns conceitos e informações resultantes do levantamento bibliográfico realizado para o desenvolvimento do trabalho. Também foram apresentados os serviços relacionados ao sistema proposto neste trabalho, tais como o Instagram, Picasa e Google Panoramio. Em um dos trabalhos relacionados, o Instagram, cada usuário tem seu álbum de fotos e cada foto é ligada ao perfil do usuário, que pode ser privado. Já no sistema proposto todas as fotos são públicas e acessíveis a qualquer pessoa que utilizá-lo. A principal diferença entre o sistema proposto e o Instagram é o público alvo e a maneira na qual as fotos são apresentadas por cada serviço. O Picasa, entretanto, tem como foco a organização do acervo de fotos localmente, faltando o aspecto de compartilhamento das fotos, que o sistema proposto tem. O sistema proposto mais se assemelha ao Google Panoramio. Entretanto, o público alvo do Google Panoramio é geral, enquanto o sistema proposto é focado em paisagens e pontos turísticos, tendo como público alvo principalmente turistas.

No próximo capítulo é apresentada a metodologia utilizada para o desenvolvimento do sistema proposto.

3 METODOLOGIA

Esta trabalho propõe o desenvolvimento de um sistema de informação que utiliza tecnologia para dispositivos móveis, focando especificamente em *smartphones*, tendo a função de aprimorar a visualização de pontos turísticos em regiões das cidades.

Para o desenvolvimento do trabalho proposto foram definidas as seguintes atividades:

- **Estudo de aplicações relacionadas ao projeto proposto**: realiza uma busca sobre as aplicações relacionadas ao presente trabalho, como citado no levantamento bibliográfico, para evitar que se dupliquem recursos desnecessários já existentes em outras aplicações e para possibilitar a comparação futura entre o protótipo proposto e as alternativas existentes;
- **Planejamento do projeto do sistema**: nesta etapa são especificados a arquitetura do projeto, o escopo e recursos a serem utilizados, a viabilidade, cronograma e os custos envolvidos em conjunto com o levantamento de riscos;
- **Análise do sistema proposto**: esta fase consiste no levantamento dos requisitos funcionais e não funcionais, modelagem da base de dados utilizando diagramas de entidade-relacionamento, modelagem e *design* do sistema utilizando U.M.L. e o detalhamento de como será a arquitetura do sistema proposto;
- **Implementação**: esta fase envolve o desenvolvimento do sistema proposto;

3.1 TECNOLOGIAS

O sistema proposto é desenvolvido com um conjunto de tecnologias que disponibilizam os recursos necessários para a implementação das funcionalidades requeridas. Tendo sido definido que a arquitetura do sistema faz uso do modelo cliente-servidor, foram definidas as tecnologias que melhor atendem às necessidades do sistema a ser desenvolvido.

Optou-se por utilizar duas tecnologias distintas para cada um dos artefatos desenvolvidos. Para o desenvolvimento da aplicação que executa no servidor foi escolhido a linguagem de programação Go em conjuto com o banco de dados PostgreSQL. Para o desenvolvimento do aplicativo móvel foi escolhido a linguagem de programação Swift em conjunto com o SDK do iOS. Seria possível a utilização de somente uma plataforma, como por exemplo Java[1], entretanto seria necessário adotar a plataforma Android para o desenvolvimento deste projeto. A escolha das tecnologias deveu-se pelos seguintes motivos (i) disponibilidade de todas as ferramentas de forma gratuita, (ii) a familiaridade do autor com as tecnologias existentes e por fim (iii) a percepção do autor que utilizando as mesmas, haverá uma diminuição da complexidade do sistema como um todo. As principais tecnologias e ferramentas que serão adotadas no presente trabalho estão detalhadas da Subseção 3.1.1 até a 3.1.8.

3.1.1 IOS SDK

O iOS SDK é o kit de desenvolvimento para iOS que provê vários *frameworks* para o desenvolvimento de aplicações para dispositivos com o sistema operacional iOS instalado. O iOS SDK é instalado em conjunto com a instalação do Xcode que está explicado na Seção 3.1.7. É possível utilizar um Simulador iOS através do próprio Xcode, onde é possível também encontrar a documentação das plataformas iOS e OS X.(APPLE, 2007).

3.1.2 LINUX

Linux é um membro da família de sistemas operacionais *Unix-like*[2]. Além do Linux, existem alguns outros sistemas operacionais *Unix-like* de código aberto, como por exemplo, o FreeBSD, NetBSD e OpenBSD. O Linux foi originalmente desenvolvido por Linus Torvalds em 1991, como um sistema operacional para computadores compatíveis com o IBM-PC baseados no microprocessador Intel 80386.

No desenvolvimento de aplicações no modelo cliente-servidor, há a necessidade de hospedagem da parte da aplicação servidor em um computador servidor. Esta parte do sistema irá fornecer serviços à aplicação cliente através do protocolo HTTP[3] *(Hypertext*

[1]Linguagem de programação de propósito geral, onde é possível desenvolver aplicações para web, desktop e dispositivos móveis.

[2]Unix-like são sistemas operacionais que se comportam de maneira similar a sistemas Unix, embora não necessariamente estejam em conformidade ou certificado a alguma versão da *Single UNIX Specification*.

[3]HTTP é um protocolo de comunicação para sistemas de informação hipermídia, distribuídos e colaborativos (BERNERS-LEE et al., 1996). Ele é a base para a comunicação de dados utilizado na web.

Transfer Protocol).

O computador servidor deve, portanto, ter um sistema operacional instalado para poder prover os serviços às aplicações servidoras. Tendo isso em vista, será utilizado o Linux em sua distribuição Ubuntu Server LTS.

3.1.3 LINGUAGEM DE PROGRAMAÇÃO GO

Go é uma linguagem de código aberto desenvolvida por um time na Google e outros contribuidores da comunidade de código aberto. Tendo sido anunciada ao público em novembro de 2009, ela é utilizada no Google e seu compilador suporta os seguintes sistemas operacionais (PIKE, 2012): Linux, Mac OS X, FreeBSD, NetBSD, OpenBSD, Plan 9 e Microsoft Windows e as arquiteturas i386, amd64 e ARM.

A linguagem é focada em concorrência, simplicidade e facilidade. É uma linguagem estaticamente tipada com uma sintaxe livremente derivada do C, com o adicional de ter os seguintes recursos: *garbage collection, type safety*, inferência de tipos, bem como uma grande biblioteca padrão (GOOGLE, 2009).

A Figura 3.1.3 exibe um fragmento da sintaxe da linguagem.

Coordenado pela World Wide Web Consortium e Internet Engineering Task Force, um dos trabalhos mais notáveis dessas duas instituições foi o RFC 2616, que definiu o protocolo HTTP/1.1.

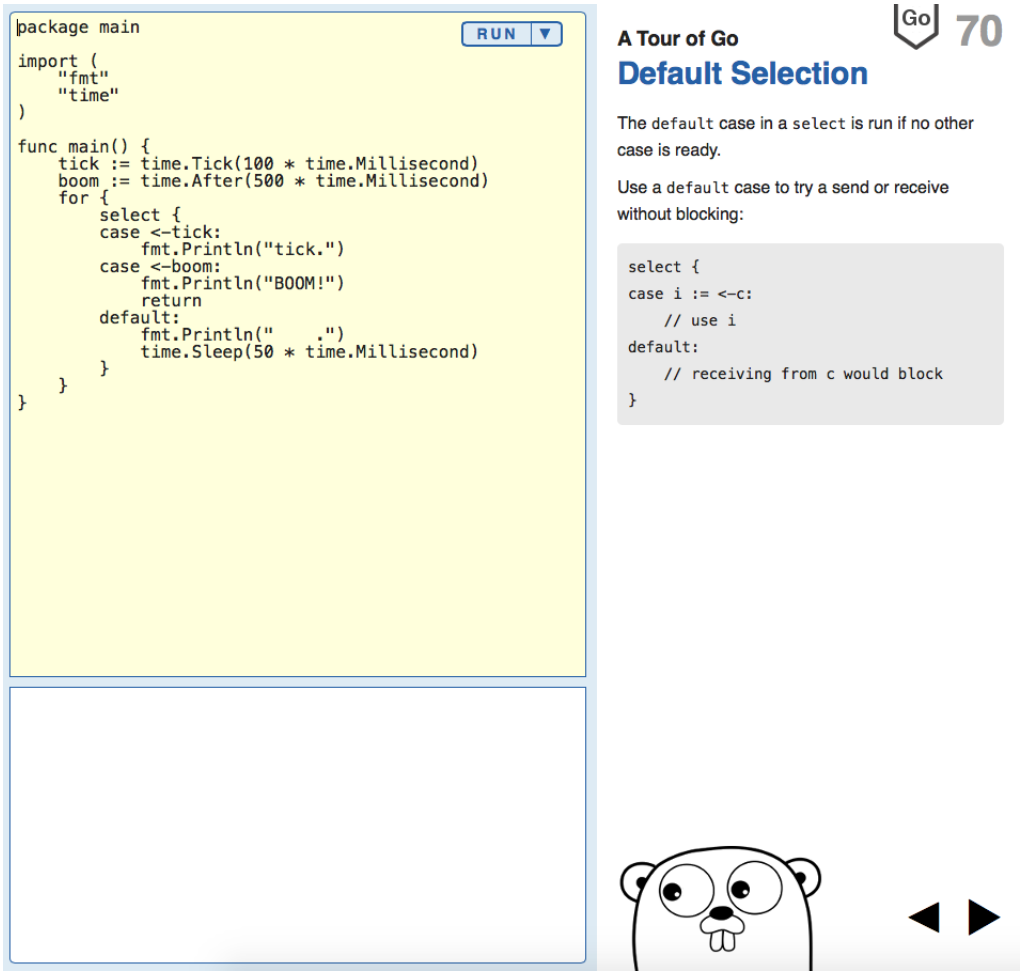

Figura 6: Exemplo de sintaxe do Tour of Go

A linguagem Go será utilizada na criação do *web service*, descrito na Seção 3.1.6, que irá prover os serviços de armazenamento das informações, acesso ao banco de dados e processamento das requisições dos usuários.

3.1.4 SWIFT

Swift é uma linguagem de programação *open source*, desenvolvida pela Apple desde 2010. Entretanto, seu desenvolvimento foi mantido em segredo, anunciada somente em 2014 na conferência da Apple *WorldWide Developers Conference* (WWDC) e disponibilizada em sua forma *open source* no dia 3 de dezembro de 2015, sob a licença *Apache License 2.0* (CONTRIBUTORS, 2015a).

Ela foi criada para substituir a linguagem de programação *Objective-C* no desenvolvimento de aplicativos para iOS, OS X, watchOS e tvOS. *Swift* é uma linguagem de programação compilada, multi-paradigma, que mistura conceitos de programação orien-

tada a objetos, programação funcional e imperativa.

Mesmo sendo uma linguagem desenvolvida pela Apple para sua própria plataforma, no lançamento da versão *open source* da linguagem também foi liberada a versão para o sistema operacional Linux (CONTRIBUTORS, 2015b). É importante ressaltar pontos importantes desse lançamento, como por exemplo:

- A versão open source não contém o *runtime* da linguagem Objective-C (CONTRIBUTORS, 2015b);
- A linguagem e suas biblioteca padrão compartilha a maioria das implementações e API com a plataforma da Apple (CONTRIBUTORS, 2015b);
- Módulo GLibc: A maioria da biblioteca C padrão do Linux está disponível através desse módulo (CONTRIBUTORS, 2015b).

A linguagem *Swift* foi utilizada na criação do protótipo do aplicativo móvel, pois é uma das linguagens para desenvolvimento de aplicativos nativo da plataforma iOS, alvo desse trabalho.

3.1.5 BANCO DE DADOS

Segundo Ullman (1997), uma base de dados nada mais é do que uma coleção de informações que existe ao longo de um período de tempo, geralmente muitos anos. Em linguagem comum, o termo *base de dados* se refere a uma coleção de dados que é gerenciada por um SGBD.

As principais funções de um SGBD são (ULLMAN, 1997):

1. Permitir que usuários possam criar novas bases de dados e especificar seus esquemas (estruturas lógicas dos dados), utilizando uma linguagem especializada de definição de dados;
2. Dar aos usuários a habilidade de buscar dados e modificá-los, utilizando a linguagem apropriada, geralmente chamada de *query language* ou *data-manipulation language*;
3. Suportar o armazenamento de uma quantidade muito grande de dados - vários terabytes ou mais - por um longo período de tempo, permitindo acesso eficiente aos dados para buscas e modificações nas bases de dados;

4. Permitir durabilidade e a recuperação dos dados em face a falhas, erros de vários tipos ou má utilização;

5. Controlar o acesso concorrente aos dados a partir de muitos usuários, sem permitir interferências inesperadas entre os usuários (propriedade de isolamento) e impedir que ações sejam realizadas parcialmente sobre os dados ao invés de completamente (propriedade de atomicidade).

No presente trabalho, será utilizado o sistema de gerenciamento de base de dados relacional PostgreSQL em conjunto com a extensão PostGIS, que são descritos a seguir.

3.1.5.1 PostgreSQL

PostgreSQL é um SGBD relacional que foca em extensibilidade e *standards-compliance*[4]. Lançado em 1995, o PostgreSQL é um banco de dados de código fonte aberto, mantido por uma comunidade de desenvolvedores de software, sendo multiplataforma o que permite que o mesmo seja executado na maioria dos sistemas operacionais *UNIX-like* e também no Windows. O PostgreSQL possui uma extensão que provê um melhor suporte para objetos geográficos chamada de PostGIS (POSTGRESQL, 2014)(POSTGIS, 2014).

A versão PostgreSQL 9.4 foi definida como sendo o SGBD a ser utilizado no sistema proposto por ser versátil, estar disponibilizado de forma aberta, ser de fácil instalação no sistema operacional definido no projeto para executar as aplicações servidoras.

3.1.6 WEB SERVICES

Segundo W3C (2004), Web Service é um sistema desenhado para suportar interações entre máquinas através de uma rede. Tem uma interface de comunicação descrita em um formato *machine-processable* especificamente *WSDL (Web Services Description Language)*.

WSDL é uma linguagem baseada no XML utilizada para descrever Web Services. Trata-se de um documento XML[5] que além de descrever o serviço, especifica como se comunicar com ele e quais as operações ou métodos disponíveis. Submetido ao W3C pelas empresas Ariba, IBM e Microsoft em março de 2001, teve seu primeiro rascunho

[4]Cumprimento de normas e padrões definidos. Neste caso, a implementação do padrão SQL pelo PostgreSQL obedece completamente as normas do padrão ANSI-SQL:2008.

[5]XML é uma linguagem de marcação que define um conjunto de regras que codificam documentos em um formato que pode ser lido tanto por máquinas quanto por humanos. É definido pela especificação W3C XML 1.0 e existem várias outras especificações relacionadas ao mesmo, que são padrões abertos (W3C, 2008).

disponibilizado em julho de 2002.

Outros sistemas interagem com o Web Service utilizando um protocolo de comunicação, como por exemplo o SOAP.[6] As mensagens trocadas entre as entidades máquinas seguem as regras do protocolo SOAP e são tipicamente transportadas sobre o protocolo HTTP com XML e em conjunto com outros padrões Web. Existem também sistemas que utilizam o padrão REST, que é uma abstração dos elementos arquiteturais dentro de um sistema de hipermídia distribuído (FIELDING, 2000). A seguir, a definição do que é o padrão REST:

> "O padrão REST ignora os detalhes de implementação de componentes e sintaxe do protocolo para focar nas regras dos componentes, as restrições entre suas interações com outros componentes, e sua interpretação de dados significantes. Ele engloba as restrições entre componentes, conectores, e dados que definem a base da arquitetura da Web, e assim, a essência do seu comportamento como uma aplicação baseada em rede (FIELDING, 2000)."

A escolha do padrão REST em detrimento do padrão SOAP ocorreu porque o REST é mais simples e mais leve para dispositivos móveis do que o SOAP.

3.1.7 XCODE

O Xcode é o ambiente de desenvolvimento de software utilizado para implementação do *frontend* do sistema proposto. Com ele será desenvolvido o aplicativo para o iOS. Esta ferramenta é desenvolvida pela Apple e dedicada ao desenvolvimento para a sua plataforma, com suporte a iOS e OS X. A mesma só funciona em computadores Apple. As linguagens suportadas pela mesma são Objective-C, Swift, C e C++ e está atualmente na sua versão 6 (APPLE, 2007).

A Figura 7 ilustra a versão 7 do XCode, onde é possível ver o *Storyboard*, local onde é possível desenhar a interface gráfica da aplicação em desenvolvimento, bem como a relação entre cada tela da aplicação.

[6] SOAP é uma especificação de protocolo para troca de informação estruturada na implementação de Web Services em redes de computadores.

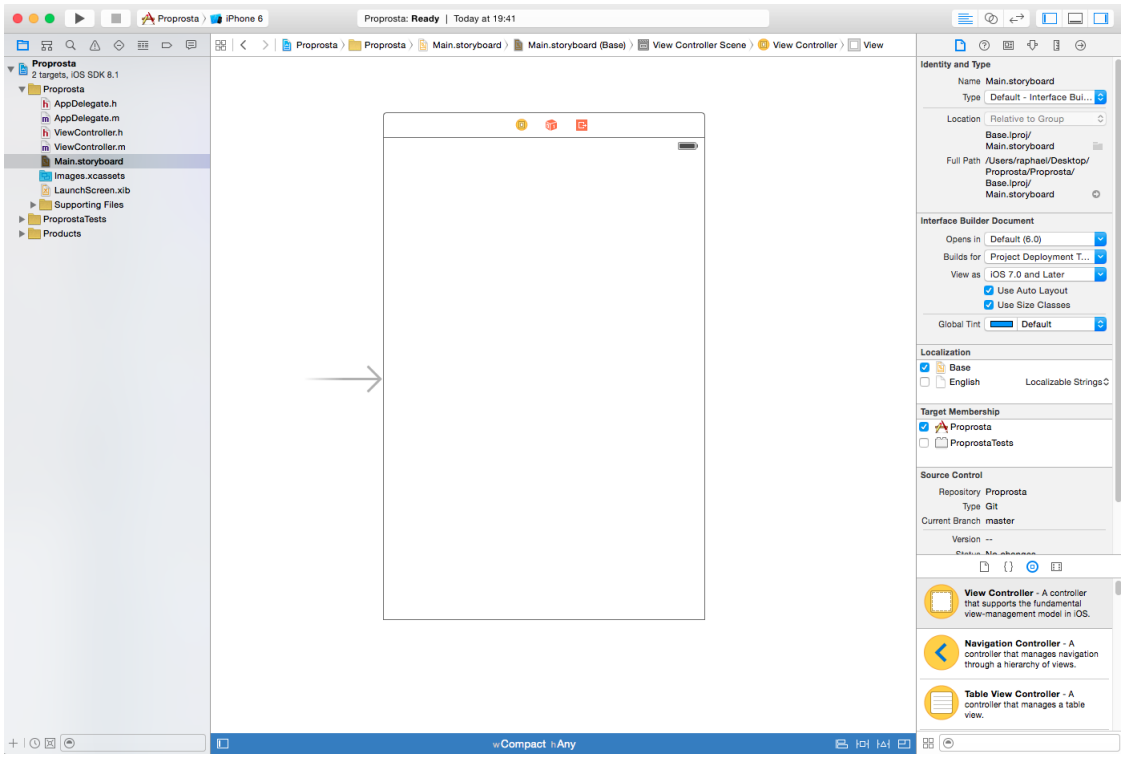

Figura 7: Xcode Storyboard.

A escolha da ferramenta Xcode foi motivada basicamente pela mesma ser focada especificamente na plataforma Apple, gratuita e por ser a ferramenta padrão para desenvolvimento de aplicações para a plataforma iOS.

3.1.8 ATOM

O Atom é utilizado para o desenvolvimento da parte *backend* do projeto. Ele é um editor de texto de código fonte aberto, desenvolvido pela empresa Github (GITHUB, 2014) com suporte a várias tecnologias através de *plugins* e com suporte à linguagem Go que será utilizada no *backend* do sistema. No momento do desenvolvimento da proposta, o Atom ainda está na versão beta.

A Figura 8 ilustra o editor de texto executando com o *plugin Go Plus*, que adiciona algumas funcionalidades ao editor de texto, podemos citar, *syntax highlight*, identação automática do código-fonte, compilação, entre outros recursos. Os fragmentos de código vistos na Figure 8 foram extraídos do protótipo.

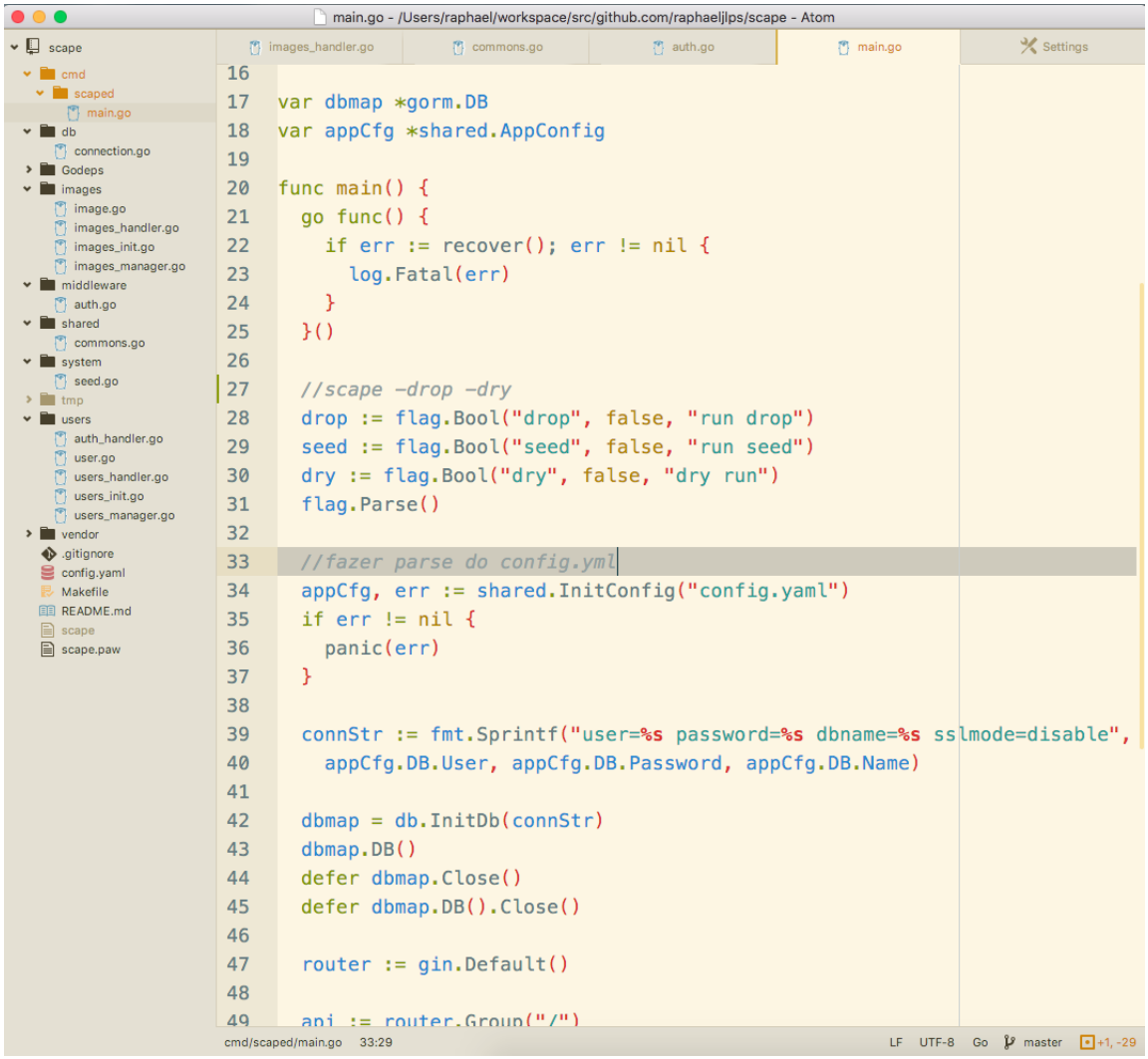

Figura 8: Editor de Texto Atom.

A escolha do Atom como uma das ferramentas de desenvolvimento foi motivada basicamente pelo fato do mesmo ter suporte à linguagem de programação Go, pelo seu bom desempenho, por ser um editor de texto leve e pela familiaridade com o mesmo.

4 RECURSOS DE HARDWARE E SOFTWARE

Este capítulo apresenta os recursos utilizados no desenvolvimento do sistema. Será mencionada também a origem dos recursos e a maneira como foram adquiridos.

4.1 RECURSOS DE HARDWARE

Para a realização do desenvolvimento e implantação do sistema proposto, foram utilizados um computador pessoal, um computador servidor para a hospedagem das imagens, da base de dados e da aplicação e um *smartphone* para a realização dos testes necessários no protótipo do sistema.

A Tabela 1 lista cada um dos recursos citados e detalha suas capacidades.

Tabela 1: Recursos de hardware

Hardware	**Processador**	**Memória**	**Armazenamento**	**Sistema Operacional**
Mac Mini	Intel Core i5	10 GB	256 GB SSD	Mac OS X 10.10
Servidor	vCPU[1]	1 GB	30 GB SSD	Ubuntu Linux 14.04 LTS
iPhone 5c	Apple A6	1 GB	8 GB Flash	iOS 8

O servidor foi alugado da empresa Digital Ocean pelo autor da proposta, o *smartphone* foi adquirido pelo autor do projeto com recursos próprios e o computador para desenvolvimento pertence ao autor da proposta.

4.2 RECURSOS DE SOFTWARE

A Tabela 2 cita os softwares utilizados para o desenvolvimento do sistema.

Tabela 2: Recursos de software

Software	Versão	Utilização
XCode	7	Desenvolvimento do aplicativo móvel
Swift	2.1	Linguagem de programação do app
iOS SDK	8	Desenvolvimento do aplicativo móvel
Atom	1	Desenvolvimento da aplicação servidor
PostgreSQL	9.4	Armazenamento dos dados
Astah	versão comunitária	criação os diagramas do sistema
Ubuntu Linux	14.04 LTS	Sistema operacional para o servidor

O Xcode foi utilizado para a implementação e desenho de interfaces do aplicativo móvel, fazendo-se necessária a utilização do iOS SDK com os seus *frameworks* disponíveis para a utilização dos recursos que o sistema operacional móvel tem a oferecer. O editor de texto Atom foi utilizado para a implementação do aplicativo servidor.

A linguagem Swift foi escolhida pelo autor por já ter familiaridade com a mesma, e, pela mesma ser a linguagem padrão para o desenvolvimento de aplicativos para iOS.

Para o servidor foi escolhido o sistema operacional Ubuntu Linux 14.04 LTS que executará a aplicação *backend*, o sistema de gerenciamento da base de dados PostgreSQL e armazenará as imagens no sistema de arquivos, pelo mesmo ser de fácil utilização, com uma facilidade de encontrar resoluções de problemas na internet e pelo autor já ter experiência na utilização do mesmo.

O Astah foi utilizado para a criação de todos os diagramas presentes neste trabalho, a escolha do mesmo se deu pelo fato de ser gratuito e o autor já ter utilizado anteriormente a ferramenta.

Todos os programas e bibliotecas citadas na tabela tem licença de código aberto, com a exceção do iOS SDK e Xcode que são disponibilizados gratuitamente pela Apple inc.

5 DESENVOLVIMENTO DO SISTEMA

Este capítulo apresenta um maior detalhamento sobre as tecnologias utilizadas no desenvolvimento do sistema, tanto do aplicativo móvel, quanto no desenvolvimento da aplicação servidor. Inicialmente é apresentada a API do iOS SDK, enfatizando os *frameworks* de Mapas e Localização. Na sequencia, é demonstrada a arquitetura de comunicação entre o aplicativo móvel e a aplicação servidor, e também a aplicação servidor. Por fim, na última seção, as telas do aplicativo móvel são descritas de maneira geral.

5.1 IOS SDK

Nesta seção são apresentados os *frameworks* que foram utilizados no aplicativo móvel, que são disponibilizados nativamente pela Apple no iOS SDK. Sendo assim, iremos discorrer os *frameworks* relativos aos Mapas - *Map Kit* (utilizado para mostrar a posição das imagens no mapa) e ao serviço de localização *Core Location* (utilizado para extrair a posição do usuário e adicionar geolocalização nas fotos).

Antes de tudo, é importante esclarecer alguns termos utilizados pela plataforma de desenvolvimento da Apple, tais como:

- **ViewController**: um objeto que gerencia um conjunto de telas e coordena o fluxo da informação entre o modelo de dados (*data model*) do aplicativo e as telas que disponibilizam aquele dado (DEVELOPERS, 2015e);
- **Delegate**: um objeto que age a favor ou em coordenação com outro objeto, quando o mesmo encontra um evento em uma aplicação. O objeto que delega é geralmente um *responder object*, isso é, um objeto que herda do objeto **UIResponder**, significando que o mesmo responde â uma ação do usuário, como por exemplo um botão. O objeto *delegate* é um objeto para qual se transfere o controle da interface do usuário para aquele evento ou é pelo menos encarregado de interpretar o evento de maneira específica para aquela aplicação (DEVELOPERS, 2015b);

- **Storyboard**: um arquivo que contém a representação visual da interface de usuário do aplicativo móvel, mostrando telas de conteúdo e a transição entre elas. Manipula-se esse arquivo utilizando o *Interface Builder* (DEVELOPERS, 2015e);

- **Segue** é uma transição entre dois *view controllers* em um *storyboard*, todas as transições de telas são feitas por meio de segues (DEVELOPERS, 2015g);

- **Data model**: é a representação ou estrutura de dados de uma aplicação (DEVELOPERS, 2015e);

- **Model-View-Controller**: é um padrão de projetos (MVC) bastante antigo. Variações do mesmo existem desde os primeiros dias do *Smalltalk*[1]. É um padrão de alto nível na medida com que se preocupa com a arquitetura global da aplicação e a classificação dos objetos, de acordo com papéis gerais que eles tomam na aplicação. É também um padrão composto, que compreende vários, mais elementares padrões. Programas orientados a objeto se beneficiam de várias maneiras quando adotam o padrão de projetos MVC para seu design. Muitos dos objetos desses programas tendem a ser mais reusáveis e suas interfaces tendem a ser melhor definidas (DEVELOPERS, 2015f);

- **Scene**: uma representação no *storyboard* do conteúdo de uma tela da aplicação. (DEVELOPERS, 2015e);

- **Event-Driven Programming**: uma categoria de programação em que o fluxo da aplicação é determinada por eventos, como por exemplo, eventos do sistema ou ações do usuário (DEVELOPERS, 2015e);

- **Auto Layout**: é um motor de *layoute* que ajuda a construir as interfaces gráficas para vários tamanhos de tela, ele utiliza as restrições especificadas pelo programador para calcular onde posicionar os componentes visuais na tela do dispositivo (DEVELOPERS, 2015e).

5.1.1 FRAMEWORK DE MAPAS

O Map Kit é um *framework* que permite embutir um mapa completamente funcional na interface das aplicações. Utilizando-o é possível dar zoom, mover-se pelo mapa, entre outras coisas, de maneira programática, exibir construções em 3 dimensões, e adicionar anotações ao mapa com informações personalizadas. O Map Kit também provê

[1]Smalltalk é uma linguagem de programação antiga, orientada a objetos, dinamicamente tipada.

suporte automático a eventos de toque que permitem ao usuário controlar o mapa (DEVELOPERS, 2015c).

Para entender como o sistema de coordenadas é usado pelo Map Kit, é interessante entender como a interface tridimensional da superfície da terra é mapeada para um mapa bidimensional. A Figura 9 mostra como a superfície da terra pode ser mapeada para uma superfície bidimensional (DEVELOPERS, 2015c).

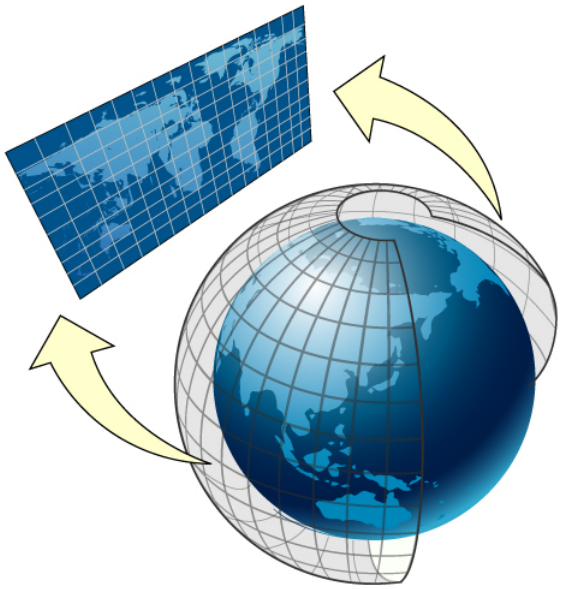

Figura 9: Mapeamento da superfície da terra para uma superfície plana.
Fonte: Developers (2015c)

O Map Kit utiliza o sistema de projeção de Mercator (DEVELOPERS, 2015c), que é um tipo de projeção cilíndrica do globo terrestre. Nessa projeção, os meridianos são planificados na forma de linhas retas paralelas verticais que são horizontalmente equidistantes, ao passo que os paralelos são planificados na forma de linhas retas paralelas horizontais de modo que a distância vertical entre dois paralelos sucessivos é tão menor quanto mais próximos esses paralelos estiverem da linha do equador (CHRISTOPHERSON, 2012). A vantagem de se utilizar esse sistema de projeção é que o conteúdo do mapa é escalado de maneira que beneficia a navegação em geral. Especificamente, na projeção de Mercator, linhas retas desenhadas entre dois pontos quaisquer produzem um curso que pode ser utilizado para navegação na superfície da terra. A projeção utilizada pelo Map Kit utiliza o meridiano primário como meridiano central (DEVELOPERS, 2015c).

O Map Kit suporta três sistemas de coordenadas básicos para especificar pontos em um mapa (DEVELOPERS, 2015c):

- Uma *map coordinate* é uma latitude e longitude em uma representação esférica da terra. *Map coordinates* são a maneira primária de especificar localizações ao redor do globo no Map Kit. É possível especificar coordenadas individuais utilizando-se de uma estrutura **CLLocationCoordinate2D**. Também é possível especificar áreas utilizando as estruturas **MKCoordinateSpan** e **MKCoordinateRegion**.

- Um *map point* é um valor x e y no sistema de projeção de Mercator. Pontos de mapa são utilizados para muitos cálculos relacionados ao Mapa ao invés de coordenadas de um mapa, pois eles simplificam a matemática envolvida nesses cálculos. Em uma aplicação, são utilizados primariamente *map points* quando se está especificando formas e posições onde há sombreamento entre formas. Se especifica um *map points* utilizando a estrutura **MKMapPoint**. É possível também especificar áreas utilizando as estruturas **MKMapSize** e **MKMapRect**.

- Um *point* é uma unidade gráfica associada com o sistema de coordenadas de um objeto de interface. *Map points* e *map coordinates* tem que ser mapeadas a pontos antes de serem desenhadas em uma interface. É possível especificar pontos individuais utilizando a estrutura **CGPoint**. Também é possível especificar áreas utilizando as estruturas **CGRect** e **CGSize**.

Na maioria das situações, o sistema de coordenadas que se é utilizado está pré-determinado pela interface do Map Kit que se está utilizando.

5.1.1.1 Convertendo entre sistemas de coordenadas

Embora tipicamente se especifique pontos em um mapa usando valores latitude e longitude, pode haver momentos em que se faça necessário converter de um para outro tipo de sistema de coordenadas. Por exemplo, geralmente usa-se *map points* quando se está especificando formas de sobreposição (DEVELOPERS, 2015c).

Existem algumas rotinas que facilitam a conversão entre os vários tipos de sistemas de coordenadas que são disponibilizados pelo iOS SDK. A maioria dessas conversões requerem um objeto de interface (*view*) pois envolvem a conversão de ou para pontos na tela (DEVELOPERS, 2015c).

Para realizar a conversão de *Map coordinates* para *Points* existem as rotinas *convertCoordinate:toPointToView:* e *convertRegion:toRectToView:*, onde a primeira rotina converte uma instância da classe *CLLocationCoordinate2D* para um ponto na tela, e a segunda rotina converte uma instância da classe *MKCoordinateRegion* para um retângulo na tela.

A conversão de *Map coordinates* para *Map points* acontece através da rotina *MKMapPointForCoordinate*, que recebe uma instância da classe *CLLocationCoordinate2D* e converte para um *MKMapPoint*.

Para converter de *Map Points* para *Map coordinates* existem as rotinas *MKCoordinateForMapPoint* e *MKCoordinateRegionForMapRect*, a primeira rotina converte uma instância da classe *MKMapPoint* para uma instância da classe *CLLocationCoordinate2D*, a segunda rotina converte uma instância da classe *MKMapPoint* para uma instância da classe *MKCoordinateRegion*.

Para realizar a conversão de *Map Points* para *Points* existem as rotinas *pointForMapPoint* que recebe como parâmetro um *MKMapPoint* e retorna um *CGPoint*, e a *rectForMapRect* que recebe como parâmetro um *MKMapRect* e retorna um *CGRect*.

Para converter um *Point* para um *Map Coordinate* existem as rotinas *convertPoint:toCoordinateFromView:* e *convertRect:toRegionFromView:*, a primeira rotina recebe um *CGPoint* como parâmetro e como resposta retorna um *CLLocationCoordinate2D*, a segunda rotina recebe um *CGRect* e como resposta retorna um *MKCoordinateRegion*.

Por fim, para converter um *Point* para um *Map Point* existem as rotinas *mapPointForPoint* que recebe como parâmetro um *CGPoint* e retorna um *MKMapPoint* e a rotina *mapRectForRect* que converte um *CGRect* para um *MKMapRect*.

5.1.2 CORE LOCATION

O *framework* Core Location permite que o aplicativo determine a localização do usuário ou a direção associada ao dispositivo. Ele utiliza do GPS disponível no dispositivo para determinar a posição e a direção do usuário. Utiliza-se as classes e protocolos deste *framework* para configurar e programar a notificação dos eventos de mudança de localização e direção do usuário. É possível também definir regiões geográficas e monitorar quando o usuário atravessa a fronteira dessas regiões (DEVELOPERS, 2015a).

O *framework* provê vários serviços que possibilitam obter e monitorar a atual localização do dispositivo (DEVELOPERS, 2015d):

- O serviço de localização *significant-change* provê uma forma que consome pouca energia, para pegar a atual localização e ser notificado quando mudanças significantes ocorrem;
- O serviço de localização padrão oferece uma forma altamente configurável de pegar

a localização e rastrear mudanças.

- O serviço de monitoramento de região permite que seja monitorado quando um usuário atravessa por determinadas regiões geográficas.

Para utilizar os recursos disponibilizados pelo Core Location, se faz necessário adicionar ao projeto do aplicativo o **CoreLocation.framework** e para acessar as classes e os cabeçalhos do *framework*, é necessário importar o <CoreLocation/CoreLocation.h> no topo dos arquivos relevantes, como por exemplo, ViewControllers que fazem uso dos recursos de geolocalização (DEVELOPERS, 2015d).

Obter os dados de localização é uma operação que consome bastante energia. Na maioria das vezes é suficiente estabelecer um posição inicial e, então, receber atualizações da posição periodicamente. Independente da importância dos dados de localização para a aplicação, deve-se escolher o serviço de localização apropriado e utilizá-lo sabiamente para evitar saturar a bateria do dispositivo do usuário (DEVELOPERS, 2015d).

5.1.2.1 Iniciando o serviço de localização padrão

O serviço de localização padrão é o jeito mais comum de obter a posição atual do usuário porque está disponível em todos os dispositivos. Antes de começar a utilizar o serviço, é necessário configura-lo especificando a exatidão dos dados localização desejada e a distância que deve ser percorrida antes que o sistema possa reportar uma nova localização. Quando se inicia um serviço, o mesmo utiliza os parâmetros especificados para determinar o hardware a ser ativado e então proceder para reportar eventos de localização. O precisão do serviço de localização padrão é necessária por aplicativos de navegação ou qualquer outro aplicativo que requer dados de localização de alta precisão. Porque este serviço tipicamente requer que o hardware de rastreamento de localização ativado por longos períodos de tempo, eles podem resultar em um alto consumo de energia (DEVELOPERS, 2015d).

Para se utilizar do serviço de localização padrão, é necessário criar uma instância da classe **CLLocationManager**, configurar as propriedades **desiredAccuracy** e **distanceFilter**. Para iniciar o recebimento das notificações de localização, é necessário designar um *delegate* a instância do **CLLocationManager** e executar o método **startUpdatingLocation**. A medida que os dados de localização se tornam disponíveis, o *location manager* (que é a instância do objeto CLLocationManager) notifica o seu objeto *delegate*. Se uma atualização de localização já foi entregue, é possível obter a localização mais recente diretamente do objeto da classe **CLLocationManager** sem precisar espe-

rar que um novo evento seja entregue. Para parar a entrega de atualizações de posição, é necessário executar o método **stopUpdatingLocation** do *location manager* (DEVELOPERS, 2015d).

Geralmente, o objetos da classe **CLLocationManager** são criados no **ViewController** que é responsável por gerenciar a localização do usuário.

O Fragmento de código 5.1 é um exemplo de como configurar um *location manager* para uso. Este fragmento do **MapViewController** implementa o protocolo **CLLocationManagerDelegate** para que o mesmo possa agir como um *delegate* para o *location manager*. A precisão configurada é **kCLLocationAccuracyBest** que é a maior possível, a o **distanceFilter** está configurado para trezentos metros. Deste modo, a cada trezentos metros serão geradas novos eventos de atualização de posição para que o MapViewController possa lidar com as mudanças de posição do usuário.

```
1
import UIKit
3 import MapKit
import CoreLocation
5
class MapViewController: UIViewController, MKMapViewDelegate,
    CLLocationManagerDelegate {
7
    @IBOutlet weak var map: MKMapView!
9
    var locationManager = CLLocationManager()
11
    override func viewDidLoad() {
13      super.viewDidLoad()

15      locationManager.delegate = self
        locationManager.desiredAccuracy = kCLLocationAccuracyBest
17      locationManager.distanceFilter = 300 //metros
        locationManager.requestWhenInUseAuthorization()
19      locationManager.startUpdatingLocation()

21  }
}
```

Fragmento de código 5.1: Configuração de um Location Manager

5.2 COMUNICAÇÃO ENTRE APLICATIVO MÓVEL E SERVIDOR

A comunicação entre o aplicativo móvel e o servidor utiliza o protocolo HTTP (Hypertext Transfer Protocol). A aplicação servidor é uma API Web que utiliza a arquitetura REST na organização dos métodos expostos ao aplicativo cliente.

A escolha do protocolo HTTP se deve à facilidade e popularidade do protocolo na implementação de APIs Web, além das funções para a manipulação do protocolo estarem incluídas na biblioteca padrão de diversas linguagens de programação. Graças a essa facilidade, tanto a implementação da parte de comunicação e transferência de dados da aplicação servidor quanto do aplicativo móvel ocorreram sem muitas complicações.

As informações que transitam entre o aplicativo cliente e a aplicação servidor são convertidas para o formato JSON (JavaScript Object Notation)[2] que age como uma linguagem que faz o intermédio entre as diferentes linguagens utilizadas no aplicativo móvel e no servidor.

Para requisitar uma imagem do servidor, o aplicativo cliente precisa enviar uma requisição HTTP do tipo GET para o *URL* https://sightscape.site/images/:id, onde :id é substituido pelo número inteiro que representa o identificador da imagem requisitada. Entretanto, é necessário estar autenticado junto ao sistema para realizar esta operação.

Para se autenticar junto ao servidor, o aplicativo cliente precisa enviar uma requisição HTTP do tipo POST com o corpo da requisição contendo um *JSON* com as credenciais do usuário, para a *URL* https://sightscape.site/auth/new. Caso as credenciais sejam válidas, o servidor irá retornar um *JSON* contendo o perfil do usuário, e um *token* de acesso para que o usuário possa realizar as próximas requisições autenticado. O *token* de acesso tem que ser adicionado ao cabeçalho das requisições das APIs Web que precisam ser autenticadas.

O fragmento de código 5.2 é um exemplo de requisição em que é necessário o cabeçalho **Authorization**. É possível ver que o mesmo é formado pelo tipo de authorização *Bearer* e o *Token* de acesso do usuário.

[2]JSON é um formato leve para troca de dados computacionais. JSON é um subconjunto da notação de objeto de JavaScript, mas seu uso não requer JavaScript exclusivamente.

```
POST /images/search HTTP/1.1
Authorization: Bearer efd273212f7f5b21447199310
Content-Type: application/json
Cookie: __cfduid=d5230948bcccf11617e7fa0f2a508f5b21447199310
Host: sightscape.site
Connection: close
User-Agent: Paw/2.2.5 (Macintosh; OS X/10.11.1) GCDHTTPRequest
Content-Length: 23

{
  "tags":
    [
    "university"
    ]
}
```

Fragmento de código 5.2: Exemplo do Authorization Header

A seguir é apresentado o software servidor, que suporta as funcionalidades de armazenamento das imagens, cadastro de usuário, autenticação e busca de imagens para a aplicação móvel.

5.3 SOFTWARE SERVIDOR

Nesta seção é apresentado o servidor, uma API Web que foi desenvolvida na linguagem de programação Go. O software servidor é responsável por gerenciar o cadastro dos usuários, a autenticação dos usuários, a busca e armazenamento das imagens, o cadastro de *tags* e o relacionamento delas com as imagens, além da busca por fotos baseada na localização do usuário.

A aplicação servidor é executada em um servidor Linux e utiliza o banco de dados PostgreSQL com a extensão PostGIS habilitada. A seguir é descrito como funciona o recebimento das fotos enviadas pelo aplicativo móvel para o servidor e onde o mesmo é armazenado.

5.3.1 RECEBENDO FOTOS DOS CLIENTES

Para enviar uma foto para a API Web, a aplicação móvel converte a imagem em um conjunto de bytes e a envia para o servidor em uma requisição HTTP. Essa requisição

contém informações a respeito da foto a ser enviada, como título e descrição no seu cabeçalho, e a foto propriamente dita, a qual é transportada no corpo da requisição.

O fragmento de código 5.3 apresenta uma requisição HTTP enviada do cliente para o endpoint /images/new, responsável pelo armazenamento de fotos submetidas pelo usuário. É possível ver os cabeçalhos específicos da aplicação servidora X-SCAPE-TITLE, X-SCAPE-DESC e X-SCAPE-TAGS, onde as informações da foto são enviadas ao servidor.

Ao receber a requisição, o servidor a redireciona para a função responsável por processá-la. É nessa função que o corpo da requisição é transformado em um arquivo e é feita a extração dos dados de localização da imagem. Caso não seja possível extrair a latitude e longitude de uma imagem, o servidor responde ao usuário com uma mensagem de erro. Após realizada a extração dos dados de localização, uma string aleatória é gerada para nomear esse arquivo de imagem que, então, é armazenado no sistema de arquivos do servidor. O caminho para o arquivo é salvo no banco de dados em conjunto com o título, descrição, latitude e longitude, além das *tags* relacionadas à imagem.

```
1  POST /images/new HTTP/1.1
   Authorization: Bearer efd273212f765fd568964ad
3  X-SCAPE-TITLE: Lazer UTFPR
   X-SCAPE-DESC: Area de lazer para os estudantes relaxarem e se reunirem para
       trocar ideias.
5  X-SCAPE-TAGS: utfpr relax landscape
   Content-Type: multipart/form-data; boundary=__X_PAW_BOUNDARY__
7  Cookie: __cfduid=d5230948bcccf11617e7fa0f2a508f5b21447199310
   Host: sightscape.site
9  Connection: close
   Content-Length: 3048571
11
   --__X_PAW_BOUNDARY__
13 Content-Disposition: form-data; name="file"; filename="IMG_0076.JPG"
   Content-Type: image/jpeg
```

Fragmento de código 5.3: Exemplo de *upload* de foto para o servidor

O fragmento de código 5.4 exemplifica uma resposta à uma requisição feita com sucesso ao endpoint /images/new, onde é retornado pelo servidor um JSON que representa a estrutura salva no banco de dados da aplicação.

```
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 18 Nov 2015 22:14:22 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 312
Connection: close
CF-RAY: 2477070e6c5e0358-MIA

{"id":2,"created_at":"2015-11-18T17:14:22.60434368-05:00","updated_at":"
    2015-11-18T17:14:22.60434368-05:00","views":0,"title":"Lazer UTFPR","
    note":"Area de lazer para os estudantes relaxarem e se reunirem para
    trocar ideias.","favs":0,"reports":0,"latitude":-25.43925,"longitude"
    :-49.26954722222222}
```

Fragmento de código 5.4: Exemplo de resposta do *upload*

A seguir é apresentada a busca de fotos com base na localização do usuário.

5.3.2 BUSCANDO FOTOS COM BASE NA LOCALIZAÇÃO

Ao iniciar o aplicativo, o usuário se depara com um mapa mostrando a sua localização atual e fotos ao seu redor. Isso ocorre pois, ao iniciar o aplicativo, o usuário envia a sua posição para o servidor, que com base nessas informações faz uma busca no banco de dados por fotos que estejam ao redor do usuário. O resultado dessa busca é retornado ao aplicativo móvel que adiciona ícones que representam as fotos no mapa do usuário.

O aplicativo faz uma requisição HTTP para http://sightscape.site/images/around/:latitude/:
como mostra o fragmento de código 5.5, onde os parâmetros latitude e longitude são substituídos pelos respectivos valores que representam a posição do usuário.

```
GET /images/around/-25.438488888888887/-49.26994722222222 HTTP/1.1
Authorization: Bearer efd273212f765fd5689
Cookie: __cfduid=d5230948bcccf11617e7fa0f2a508f5b21447199310
Host: sightscape.site
Connection: close
User-Agent: Paw/2.2.5 (Macintosh; OS X/10.11.1) GCDHTTPRequest
```

Fragmento de código 5.5: Exemplo de requisição para a busca de foto com base na localização do usuário

É possível ver no fragmento de código 5.6 o método create. É através desse método que as imagens são armazenadas no sistema de arquivos do servidor e no seu banco de dados. Pode-se ver na linha 2 que o conteúdo da requisição está sendo transformado em uma estrutura do tipo multipartFile. Posteriormente, são lidos do cabeçalho o título, as notas e as tags associadas à imagem. Na linha 19 pode-se ver a decodificação desse arquivo. Caso nenhum erro tenha ocorrido, na linha 25 o método LatLong retorna a latitude e longitude decodificada dos dados EXIF do arquivo recebido. Vale notar a linha 38 pois é nela que o arquivo da imagem é armazenado no sistema de arquivos do servidor. Por fim, os dados de localização do arquivo no sistema, latitude, longitude, título, entre outros, são armazenados no banco de dados (linha 51).

```
func (handler *Handler) create(c *gin.Context) {
  multipartFile, header, err := c.Request.FormFile("file")
  title := c.Request.Header.Get("X-SCAPE-TITLE")
  notes := c.Request.Header.Get("X-SCAPE-NOTE")
  tagsParam := c.Request.Header.Get("X-SCAPE-TAGS")
  tags := strings.Split(tagsParam, " ")
  if err != nil {
    c.JSON(http.StatusBadRequest, shared.Response{Error: &shared.Error{err.
    Error(), 1}})
    return
  }
  defer multipartFile.Close()

  file, err := ioutil.ReadAll(multipartFile)
  if err != nil {
    c.JSON(http.StatusBadRequest, shared.Response{Error: &shared.Error{err.
    Error(), 1}})
    return
  }
  reader := bytes.NewReader(file)
  x, err := exif.Decode(reader)
  if err != nil {
    c.JSON(http.StatusInternalServerError, shared.Response{Error: &shared.
    Error{"couldn't decode your file", 1}})
    return
  }

  lat, lng, err := x.LatLong()
```

```
26    if err != nil {
        c.JSON(http.StatusInternalServerError, shared.Response{Error: &shared.
      Error{"couldn't read your photo longitude and latitude", 1}})
28      return
      }
30
      u, exists := c.Get(shared.AuthUserKey)
32    if !exists {
        c.JSON(http.StatusInternalServerError, shared.Response{Error: &shared.
      Error{"internal server error", 1}})
34      return
      }
36
      var filePath string
38    if filePath, err = shared.SaveImage(file, handler.c.Server.PhotosDir,
      header.Filename); err != nil {
        c.JSON(http.StatusInternalServerError, shared.Response{Error: &shared.
      Error{err.Error(), 1}})
40      return
      }
42
      i := Image{}
44    i.UserID = uint(u.(int64))
      i.Path = filePath
46    i.Latitude = lat
      i.Longitude = lng
48    i.Point = gormGIS.GeoPoint{Lat: lat, Lng: lng}
      i.Title = title
50    i.Note = notes
      if err = handler.m.SaveImage(&i, tags); err != nil {
52      c.JSON(http.StatusInternalServerError, shared.Response{Error: &shared.
      Error{err.Error(), 1}})
        return
54    }

56    c.JSON(http.StatusOK, i)
  }
```

Fragmento de código 5.6: Reprodução do médoto que salva as imagens no servidor

Na próxima seção é apresentada a interface gráfica, onde são apresentadas as telas que compõem o aplicativo móvel.

5.4 INTERFACE GRÁFICA

Foi desenvolvida uma interface gráfica simples e limpa para o aplicativo móvel, que pudesse prover todas as funcionalidades especificadas com fácil acesso e que o fluxo de navegação na aplicação móvel seja simples e prático. A interface foi desenvolvida dessa maneira para que o usuário final não tenha problemas na utilização do aplicativo, tanto na hora de buscar uma imagem ou submeter uma imagem ao serviço, quanto na hora de navegar pelo mapa de imagens. A interface gráfica é composta de várias partes, cada uma dessas partes é implementada em sua própria classe ViewController do iOS. Na sequência, são apresentadas as telas do aplicativo.

5.4.1 TELA INICIAL

A primeira tela apresentada ao usuário é a Inicial, é a partir dela que o usuário irá se autenticar no sistema e poder navegar pelas outras telas. A tela inicial apresenta duas opções ao usuário, que é a de efetuar *Log in* caso o mesmo já tenha feito o seu cadastro previamente ou, de *Sign up* caso seja a primeira vez que o usuário esteja utilizando o sistema.

Para se autenticar ou fazer o seu cadastro é necessário ter um endereço de e-mail e uma senha. Quando o usuário toca no botão *Log in*, o aplicativo envia uma requisição ao servidor com as credenciais que o usuário digitou nos campos de e-mail e password. Caso haja sucesso na autenticação, o servidor irá responder com o perfil do usuário, que entre outras coisas, tem o *token* de acesso do usuário.

A Figura 10 ilustra a tela Inicial que o usuário irá encontrar ao executar o aplicativo móvel no seu *smartphone*. Nela, é possível ver o *banner* do aplicativo, os campos de texto de e-mail e senha e também os botões de *log in* e *sign up*.

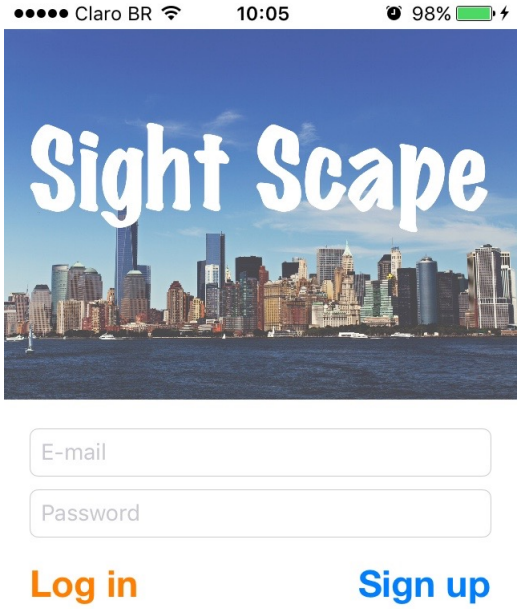

Figura 10: Tela Inicial

A seguir é apresentado a tela do mapa e suas variações, que é a primeira tela a ser vista pelo usuário após a tela Inicial.

5.4.2 TELA DO MAPA

A segunda tela do aplicativo, e uma das mais importantes, é a tela do Mapa (ver Figura 11. É nela que o usuário localiza sua posição atual e também a posição das fotos disponibilizadas pelo sistema na área ao seu redor. Na tela do mapa é possível visualizar as outras telas disponíveis para o usuário pelo aplicativo, como a tela de *search* (ícone de lupa) que o usuário utiliza para buscar por fotos, ou a tela de *photos* (ícone de câmera fotográfica) que o usuário utiliza para selecionar e enviar uma foto para o serviço. Por último, temos a tela de *Profile* (ícone do usuário), onde o usuário pode visualizar as informações do seu perfil.

Na tela de Mapas, o círculo azul com borda branca representa a posição atual do usuário. Vale lembrar que nem sempre a posição será exata. Caso o usuário clique

no círculo azul, uma *popup* irá aparecer exibindo a seguinte frase "*Current Location*". Entretanto, se o usuário selecionar um dos pinos vermelhos no mapa, uma *popup* irá surgir para exibir o título e descrição da foto, ambos escolhidas pelo usuário que enviou a foto para o serviço. Contudo, antes de exibir a tela de detalhes da foto, o aplicativo móvel verifica se a foto selecionada está disponível localmente para poder exibi-la. Caso a foto não esteja disponível, o aplicativo então requisita a foto ao servidor. O servidor, então, retorna a foto para o aplicativo, que a armazena no álbum de fotos do dispositivo e a exibe na tela.

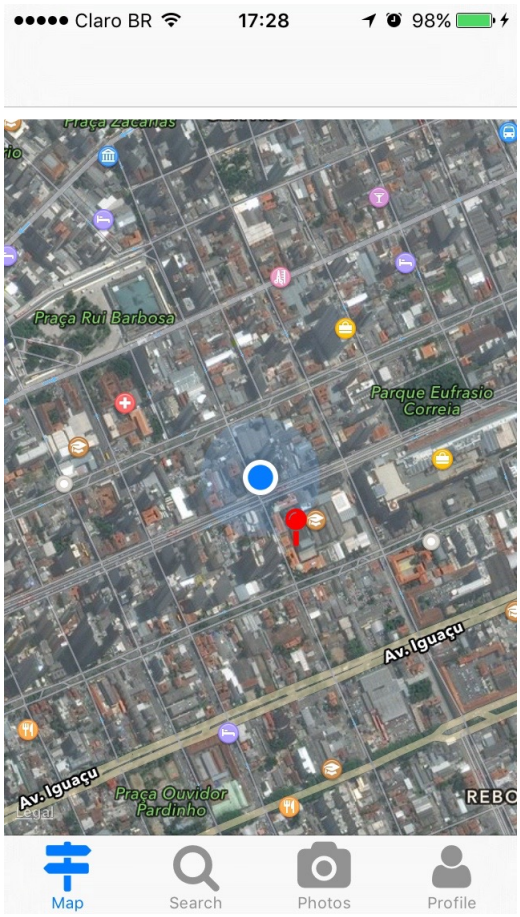

Figura 11: Tela do Mapa.

A Figura 12 ilustra a tela de Mapa quando o usuário seleciona um dos pinos e uma *popup* aparece com o título e a descrição da foto. Na figura é possível ver também o símbolo de informação da cor azul. Caso o usuário deseje visualizar a foto, ele precisa selecionar este símbolo que o levará a tela de detalhes da foto como mostra a Figura 13.

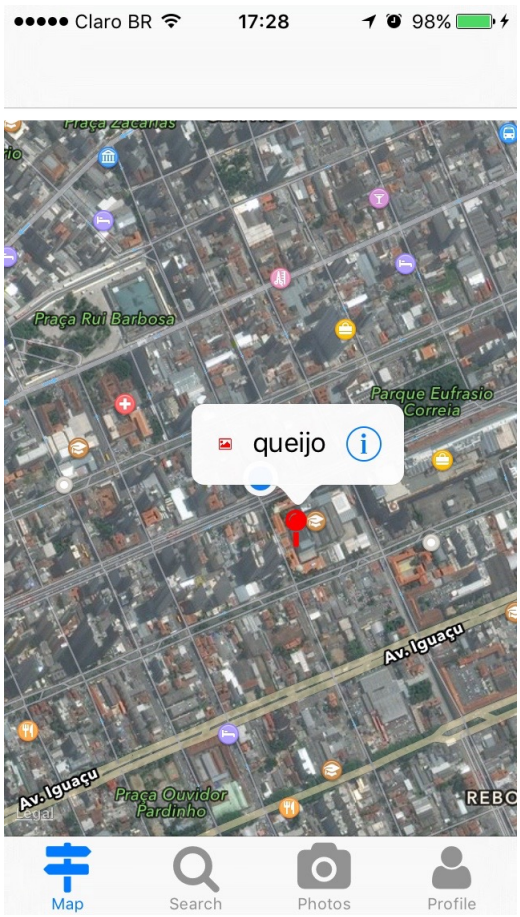

Figura 12: Tela do Mapa com imagem selecionada.

A Figura 13 exibe a tela de detalhes da foto onde é possível ver o título da foto no topo da figura, a foto e os botões de *Favorite* e *Report abuse*, utilizados para indicar a foto como favorita e reportar abuso, respectivamente. Há duas maneiras de se chegar na tela de detalhes de foto: a primeira e mais comum é através da seleção de uma foto no mapa e a segunda é através da tela de buscas. Uma vez que o usuário seleciona o botão de reportar um abuso, o aplicativo móvel faz uma requisição à aplicação servidor informando o identificador da foto reportada.

Figura 13: Tela de detalhes da foto.

A seguir é apresentada a tela de busca fotos, onde é possível buscar uma foto através de *tags*.

5.4.3 TELA DE BUSCA DE FOTOS

Na tela de busca de fotos (ver Figura 14) é possível digitar *tags* para buscar uma foto no servidor (campo de busca). Quando o usuário seleciona o botão buscar, o aplicativo móvel envia uma requisição ao servidor com as *tags* informadas pelo usuário, que então retorna com uma lista de fotos. O usuário poderá, então, selecionar uma foto que deseje visualizar. Ao selecionar o ícone de informação, o usuário será direcionado para a tela de detalhes da foto. Se não houver nenhuma foto marcada com as *tags* digitadas pelo usuário, o servidor então retornará uma lista vazia.

Caso não haja internet disponível no dispositivo no momento da busca, o aplicativo informa o usuário para tentar novamente mais tarde, quando a conexão estiver restaurada.

A Figura 14 ainda exibe o botão cancelar (”*cancel*”) e a lista de itens, com o item “queijo”e o botão de informação.

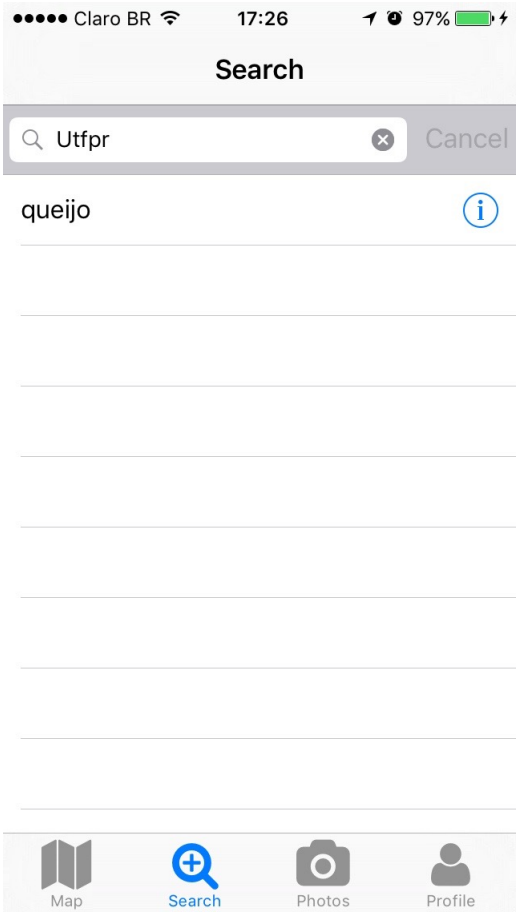

Figura 14: Tela de busca de fotos.

A seguir é apresentada a tela de fotos, local onde é possível selecionar uma foto do álbum de fotos do dispositivo e enviar para o serviço.

5.4.4 TELA DE FOTOS

É na tela de fotos (ver Figura 15 que o usuário consegue escolher uma foto para disponibilizar no serviço. Primeiramente, ele precisa selecionar a foto clicando no botão em formato de câmera que está no canto superior direito da tela. Ao fazer isso, uma tela de seleção de fotos que estão no seu álbum de fotos irá aparecer. Após a foto ter sido selecionada, o usuário irá ver uma tela como a da Figura 16, onde a foto selecionada aparece no centro da tela. Caso a foto selecionada não tenha nenhuma informação de localização

embutida, o aplicativo informa ao usuário que não é possível enviá-la ao servidor e pede ao usuário que escolha outra foto.

Figura 15: Tela de fotos.

Figura 16: Tela de fotos com imagem selecionada.

Após selecionada uma foto com dados de localização embutidos, o botão de envio de foto, representado pelo ícone de nuvem com uma seta apontando para cima, é habilitado, permitindo que o usuário envie a foto para o servidor. Ao tocar no botão de envio de foto, o usuário é redirecionado para a tela de envio de foto que será explicada na próxima subseção.

5.4.4.1 Tela de Envio de Foto

Assim que o usuário tocar no botão de envio de foto da tela de fotos que foi discutida anteriormente na subseção 5.4.4, ele será direcionado para a tela de envio de fotos (ver Figura 17) onde será apresentado o formulário para preenchimento das informações sobre a foto.

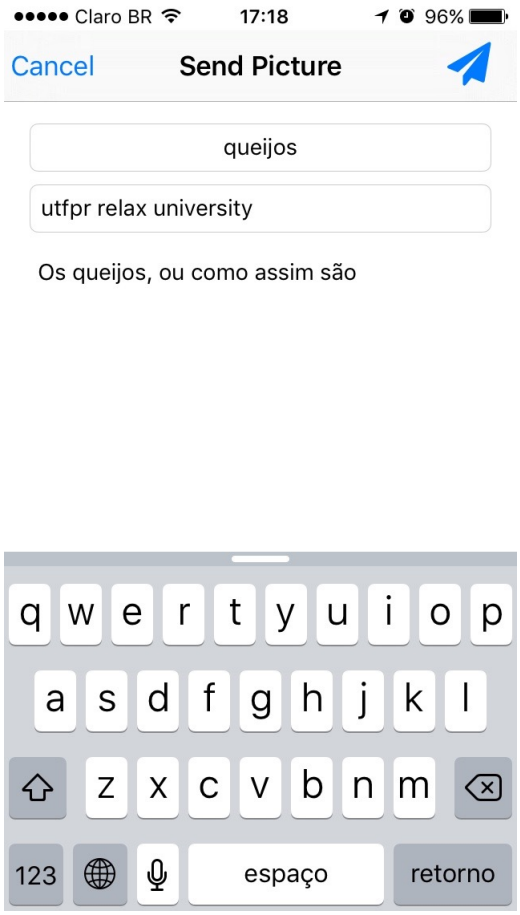

Figura 17: Tela do formulário de envio de foto.

O primeiro campo de texto disposto na tela é o do título da foto, seguido pelo campo de texto para adição de *tags*, que devem ser inseridas separadas por espaços em branco e, por fim, temos o campo de texto para a descrição ou notas sobre a foto. Na Figura 17 é possível ver um exemplo de preenchimento do formulário, com o título "queijos"e as três *tags* "utfpr relax university", além do inicio da descrição da foto. O usuário é obrigado a inserir o título na foto para poder enviá-la.

Ao tocar no botão enviar, o usuário é retornado à tela de fotos, que estará com um ícone indicando que o envio está sendo feito (indicador de atividade), como pode ser visto na Figura 18, além de ter o título da tela alterado para "*Sending Photo*". Após o envio da imagem, o título da tela volta a ser "Photos", o indicador de atividade desaparece e a imagem é removida da tela.

Figura 18: Tela exibida enquanto a foto está sendo enviada.

A seguir é apresentada a tela de perfil do usuário, onde é possível ver as informações relacionadas ao usuário como, por exemplo, a quantidade de fotos enviadas ao serviço, nome, e-mail e biografia.

5.4.5 TELA DE PERFIL DO USUÁRIO

Por último, tem-se a tela de perfil do usuário, ilustrada na Figura Figura 19. Nesta tela é possível ver as informações cadastradas, como e-mail, nome, biografia, entre outras. Como as informações disponibilizadas no perfil não são obrigatórias para o cadastro, exceto o e-mail, há uma opção de edição do perfil onde o usuário pode preenchê-las. Na tela de perfil, o usuário também pode fazer o *"log out"* do aplicativo, pois quando ele realiza o *"log in"* suas informações ficam armazenadas no aplicativo móvel e toda vez que o mesmo for executado, ele utiliza as credenciais armazenadas para fazer as requisições ao aplicativo servidor.

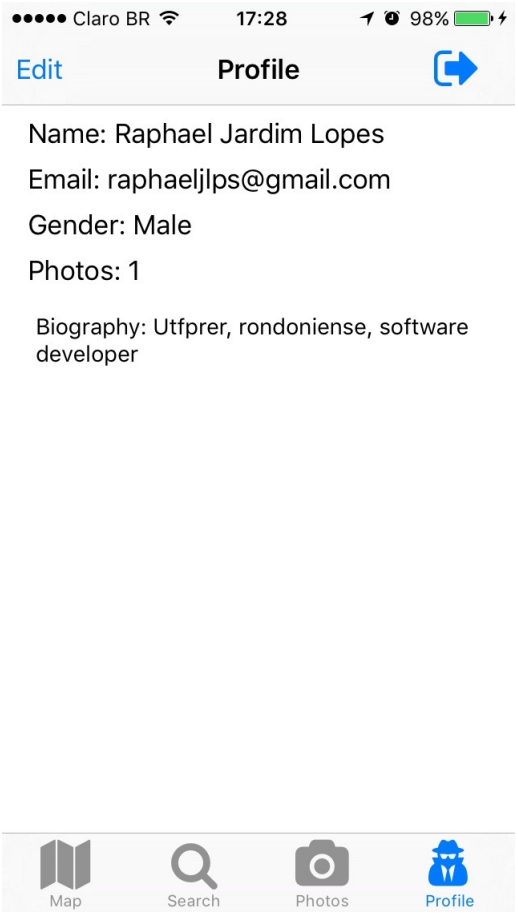

Figura 19: Tela do perfil do usuário.

A Figure 20 ilustra a tela de edição do perfil do usuário, onde é possível alterar as informações cadastradas no perfil do usuário. Nesta tela é possível identificar os campos para a alteração de nome, sexo e biografia. Há um botão para voltar a tela de perfil do usuário no canto superior esquerdo e um botão para salvar as alterações no canto superior direito. Quando o usuário toca no botão de salvar, o aplicativo móvel envia uma requisição ao servidor informando as alterações que precisam ser persistidas na base de dados.

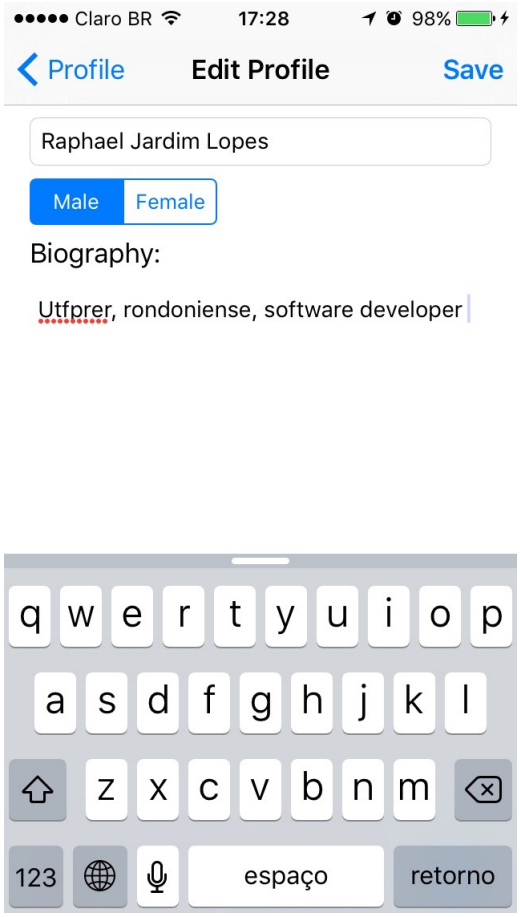

Figura 20: Tela de edição do perfil do usuário.

No próximo capítulo é apresentada a conclusão do trabalho.

6 CONCLUSÃO

Neste trabalho de conclusão de curso foi apresentada a proposta para o desenvolvimento de um sistema de compartilhamento de imagens geolocalizadas por meio de dispositivos móveis. A queda nos preços dos *smartphones* e a popularização da internet móvel tornaram esses dispositivos parte do dia a dia das pessoas. Esses aparelhos deram as pessoas a capacidade de acessar informações quase que instantaneamente e de onde estiverem, graças à cobertura das redes de telefonia móvel e sem fio.

A predominância desses dispositivos facilitou também a troca de informação entre as pessoas, surgindo um fenômeno conhecido como *crowdsourcing*. O *crowdsourcing* ocorre quando as informações que alimentam os sistemas de informação são produzidas pela comunidade de utilizadores do sistema. E por que não aproveitar desse fenômeno trocar informações a respeito de locais interessantes? Foi isso que este trabalho se propôs a fazer. Utilizando os recursos de comunicação e geolocalização disponíveis hoje em dia na maioria dos *smartphones* modernos, o sistema cria uma rede de troca de imagens e dicas sobre locais entre os seus usuários.

A partir da criação da proposta do sistema, foi realizada uma análise de possíveis soluções disponíveis para o desenvolvimento do protótipo. Essa análise também ajudou a justificar as escolhas feitas para desenvolver o trabalho. Após realizada a análise, foi criada a metodologia que seria utilizada para organizar o trabalho no decorrer de seu desenvolvimento. Foi também realizado um levantamento dos requisitos e dos casos de uso para o sistema.

Como fruto desta pesquisa tecnológica, obteve-se um protótipo de um sistema de compartilhamento de imagens que, a partir da comunicação com um servidor web, atualiza os dados de locais para usuários do sistema com base na localização atual do mesmo. A utilização da tecnologia móvel possibilitou atingir os objetivos deste trabalho, pois com ela foi possível obter com certa precisão a localização do usuário e a extração de dados de localização das fotos submetidas pela comunidade de utilizadores.

É possível concluir que a utilidade desse sistema possui limitações. É necessário que hajam muitos utilizadores para preencher o sistema com fotos.

Um obstáculo apareceu durante o desenvolvimento do sistema proposto: o envio das fotos com os metadados de localização. A maneira usual de se enviar uma foto no iOS para um servidor na web, é, após selecionada a imagem, a transformá-la em um conjunto de bytes para enviar na requisição que é feita para o servidor. Entretanto, ao selecionar a imagem utilizando um componente de alto nível, percebeu-se que os dados de localização não estavam sendo carregados junto da imagem. A partir da constatação anterior, foi necessário estudar o *framework* de manipulação de *Assets* disponível no SDK do iOS em um nível de abstração mais baixo, em relação a maneira usual de seleção e carregamento da imagem. O problema foi resolvido fazendo uma junção entre os dois componentes, utilizando o componente de alto nível para permitir que o usuário escolhe-se a imagem de seu álbum de fotos e o *framework* de *Assets* para carregar a imagem e prepara-lá para ser enviada ao servidor.

Estima-se que o desenvolvimento total do sistema, levando em conta tanto o desenvolvimento da aplicação servidor quanto do aplicativo móvel, tenha levado em torno de 630 horas de trabalho.

No que diz respeito aos trabalhos futuros, algumas melhorias ao sistema como um todo são expostas na seção 6.1.

6.1 TRABALHOS FUTUROS

O sistema apresentado neste Trabalho de Conclusão de Curso é um protótipo do que pode se tornar um sistema completo. Portanto, diversas melhorias podem ser feitas para aumentar a qualidade e eficiência do sistema. Dentre as principais melhorias destacam-se:

- Criar um formulário detalhado para denúncias de fotos. Esse detalhamento pode ajudar na tomada de análise da denúncia;
- Melhorar a apresentação das mensagem de erro que a aplicação servidor retorna para a aplicação móvel, possivelmente adicionando códigos de status em conjunto com cada mensagem de erro;
- Para atingir um público alvo maior, seria necessário criar um aplicativo móvel para Android;

- Melhorar a maneira com a qual o aplicativo móvel lida com a indisponibilidade de internet;
- Desenvolver um painel administrativo para a aplicação servidor, para realização de análises de utilização, verificação de denúncias e outras medidas administrativas.

REFERÊNCIAS

APPLE. **iOS SDK**. 2007. Disponível em: <https://developer.apple.com>. Acesso em: 23 de Novembro de 2014.

APPLE. **App Store, Categoria de Viagens**. outubro 2014. Disponível em: <https://itunes.apple.com/br/genre/ios-viagens/id6003?mt=8>. Acesso em: 16 de Novembro de 2014.

APPLE. **iOS Technology overview**. 2014. Disponível em: <http://migre.me/n4Eg7>. Acesso em: 23 de Novembro de 2014.

BARCZYSZYN, G. L.; LOPES, R. J. Comparação entre métodos de indexação utilizando folksonomia e crawling. 2013.

BERNERS-LEE, T.; FIELDING, R.; FRYSTYK, H. **Hypertext Transfer Protocol – HTTP/1.0**. Maio 1996.

CÂMARA, G.; MONTEIRO, A. M. V. Conceitos básicos em ciência da geoinformação. **Câmara G, Davis C, Monteiro AMV, organizadores. Introdução à ciência da geoinformação.[acessado 2005 Out 19]. Disponível em: http://www. dpi. inpe. br/gilberto/livro/introd[Links]**, 2004.

CAMERA, S. of the; ASSOCIATION, I. P. **Exchangeable image file format for digital still cameras: Exif Version 2.3**. 2012. Disponível em: <http://goo.gl/cPzMP6>. Acesso em: 03 de Fevereiro de 2015.

CHAIANUCHITTRAKUL, C. **Crowdsourcing Privacy Policy Analysis: Evaluating the Comfort, Readability and Importance of Privacy Policies**. Dezembro 2013. Disponível em: <http://goo.gl/aUYt4d>. Acesso em: 03 de Fevereiro de 2015.

CHRISTOPHERSON, R. **Geossistemas - 7.ed.: Uma Introdução à Geografia Física**. Bookman, 2012. ISBN 9788540701069. Disponível em: <https://books.google.com.br/books?id=kWA_P09nLa8C>. Acesso em: 1 de Dezembro de 2014.

COMPANY, A. A. analytics. **App Annie Index: Market Q1 2015**. 2015. Disponível em: <http://blog.appannie.com/app-annie-index-market-q1-2015>. Acesso em: 15 de Novembro de 2015.

CONTRIBUTORS, S. L. **Swift Language**. 2015. Disponível em: <http://www.swift.org>. Acesso em: 10 de Dezembro de 2015.

CONTRIBUTORS, S. L. **Swift Language on Linux**. 2015. Disponível em: <https://swift.org/blog/swift-linux-port/>. Acesso em: 11 de Dezembro de 2015.

DAVINCI1993. **Tela principal do iOS 7**. 2014. Disponível em: <http://migre.me/n4CGu>. Acesso em: 23 de Novembro de 2014.

DEVELOPERS, A. **Core Location Framework Reference**. 2015. Disponível em: <https://goo.gl/nmjR3B>. Acesso em: 13 de Outubro de 2015.

DEVELOPERS, A. **Delegates and Data Sources**. 2015. Disponível em: <https://goo.gl/NxLQYi>. Acesso em: 1 de Setembro de 2015.

DEVELOPERS, A. **Displaying Maps**. 2015. Disponível em: <https://goo.gl/MdHA7a>. Acesso em: 10 de Outubro de 2015.

DEVELOPERS, A. **Getting the User's Location**. 2015. Disponível em: <https://goo.gl/nrdp31>. Acesso em: 13 de Outubro de 2015.

DEVELOPERS, A. **iOS Glossary**. 2015.

DEVELOPERS, A. **Model-View-Controller**. 2015. Disponível em: <https://goo.gl/06wJE2>. Acesso em: 21 de Agosto de 2015.

DEVELOPERS, A. **Using Segues**. 2015. Disponível em: <https://goo.gl/ouJWq6>. Acesso em: 1 de Outubro de 2015.

DIGITAL, F. **iMapflickr flickr mashups and photo gallerys for your website.** 2014. Disponível em: <http://imapflickr.com>. Acesso em: 11 de Novembro de 2014.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. Tese (Doutorado), 2000. AAI9980887.

FLICKR. **Flickr Map**. 2015. Disponível em: <https://www.flickr.com/map>. Acesso em: 4 de Fevereiro de 2015.

GITHUB. **Atom text editor**. 2014. Disponível em: <http://atom.io>. Acesso em: 23 de Novembro de 2014.

GOLDER, S. A.; HUBERMAN, B. A. The structure of collaborative tagging systems. 2006.

GOOGLE. **Go language project**. novembro 2009. Disponível em: <https://golang.org>. Acesso em: 23 de Novembro de 2014.

GOOGLE. **Google Play Store, Categoria Turismo e Local**. outubro 2014. Disponível em: <http://migre.me/n4LmD>. Acesso em: 16 de Novembro de 2014.

GOOGLE. **Picasa and Picasa Web Albums help**. 2014. Disponível em: <https://support.google.com/picasa/answer/43897?hl=en>. Acesso em: 1 de Dezembro de 2014.

HOCHMAN, N.; SCHWARTZ, R. Visualizing instagram: Tracing cultural visual rhythms. In: **Proceedings of the Workshop on Social Media Visualization (SocMedVis) in conjunction with the Sixth International AAAI Conference on Weblogs and Social Media (ICWSM–12)**. [S.l.: s.n.], 2012. p. 6–9.

HOWE, J.; ROBINSON, M. **Crowdsourcing: A Definition**. Junho 2006. Disponível em: <http://goo.gl/A82Zec>. Acesso em: 3 de Fevereiro de 2015.

INSTAGRAM. **Mapa de fotos - Central de ajuda do Instagram**. 2014. Disponível em: <https://help.instagram.com/169549819835551>. Acesso em: 5 de Novembro de 2014.

LASTFM. **Last.FM**. 2015. Disponível em: <http://www.lastfm.com.br/about>. Acesso em: 4 de Fevereiro de 2015.

LI, L. T.; TORRES, R. da S. [S.l.]: Morgan and Claypool Publishers., 2014. 63-117 p.

MOURA, A. C. M. **Geoprocessamento na gestão e planejamento urbano**. [S.l.]: A Autora, 2003.

PIKE, R. **Go at Google**. outubro 2012. Disponível em: <https://talks.golang.org/2012/splash.slide>. Acesso em: 23 de Novembro de 2014.

PIRAQUIVE, F. N. D.; AGUILAR, L. J.; GARCíA, V. H. M. Taxonomía ontología y folksonomía qué son y qué beneficios u oportunidades presentan para los usuarios de la web? 2009.

POSTGIS. **PostGIS project**. 2014. Disponível em: <http://postgis.net>. Acesso em: 24 de Novembro de 2014.

POSTGRESQL. **PostgreSQL project**. 2014. Disponível em: <http://www.postgresql.org/about/>. Acesso em: 24 de Novembro de 2014.

THEOHARIDOU, M.; MYLONAS, A.; GRITZALIS, D. A risk assessment method for smartphones. In: GRITZALIS, D.; FURNELL, S.; THEOHARIDOU, M. (Ed.). **Information Security and Privacy Research**. [S.l.]: Springer Berlin Heidelberg, 2012, (IFIP Advances in Information and Communication Technology, v. 376). p. 443–456. ISBN 978-3-642-30435-4.

ULLMAN, J. **First course in database systems**. [S.l.]: Prentice Hall Inc., 1997. Tradução própria. ISBN 0-13-861337-0.

W3C. **Web Services Glossary**. fevereiro 2004. Tradução própria. Disponível em: <http://www.w3.org/TR/ws-gloss/>. Acesso em: 24 de Novembro de 2014.

W3C. **Especificação W3C XML 1.0**. Novembro 2008. Disponível em: <http://www.w3.org/TR/REC-xml>. Acesso em: 4 de Fevereiro de 2015.

APÊNDICE A – PROJETO DE SOFTWARE

Neste capítulo o projeto de *software* é apresentado. Iniciando pelo levantamento dos requisitos, onde são enumeradas algumas das funções do sistema, seguido pela definição dos casos de uso, tomando como base os requisitos levantados.

A.1 LEVANTAMENTO DE REQUISITOS

Essa seção apresenta os requisitos funcionais e não funcionais do sistema.

A.1.1 REQUISITOS FUNCIONAIS

Os requisitos funcionais levantados para o sistema são:

1. O sistema deverá permitir o cadastro de usuários.
2. O sistema deverá permitir que os usuários se autentiquem no sistema.
3. O sistema deverá permitir o envio de imagens geo-localizadas por um dispositivo móvel.
4. O sistema deverá permitir a adição de palavras-chave em uma imagem.
5. O sistema deverá permitir a adição de notas em uma determinada imagem.
6. O sistema deverá permitir visualizar as imagens que estão próximas da região onde o dispositivo se encontra tanto em um mapa, como também em uma tela de ampliação de imagem.
7. O sistema deverá conter um mecanismo que permita a busca de imagens por (*tags*).
8. O sistema deverá permitir que um usuário denuncie uma imagem imprópria, de propaganda, ofensiva, etc.

A.1.2 REQUISITOS NÃO FUNCIONAIS

Os requisitos não funcionais levantados para o sistema são:

1. O aplicativo móvel deve executar em um *smartphone* com o sistema operacional mínimo iOS 8.
2. O aplicativo móvel deve utilizar a tecnologia Wireless ou 3G/4G para funcionar com toda a sua capacidade.
3. O sistema deve utilizar de uma arquitetura de rede cliente-servidor.
4. O aplicativo móvel deve utilizar a linguagem de programação Swift.
5. O aplicativo servidor deve utilizar a linguagem de programação Go.
6. O aplicativo servidor deve executar sobre o sistema operacional Linux.

A.2 CASOS DE USO

Nesta seção os casos de uso do sistema são apresentados (ver Figura 21).

A.2.1 DIAGRAMA DE CASOS DE USO

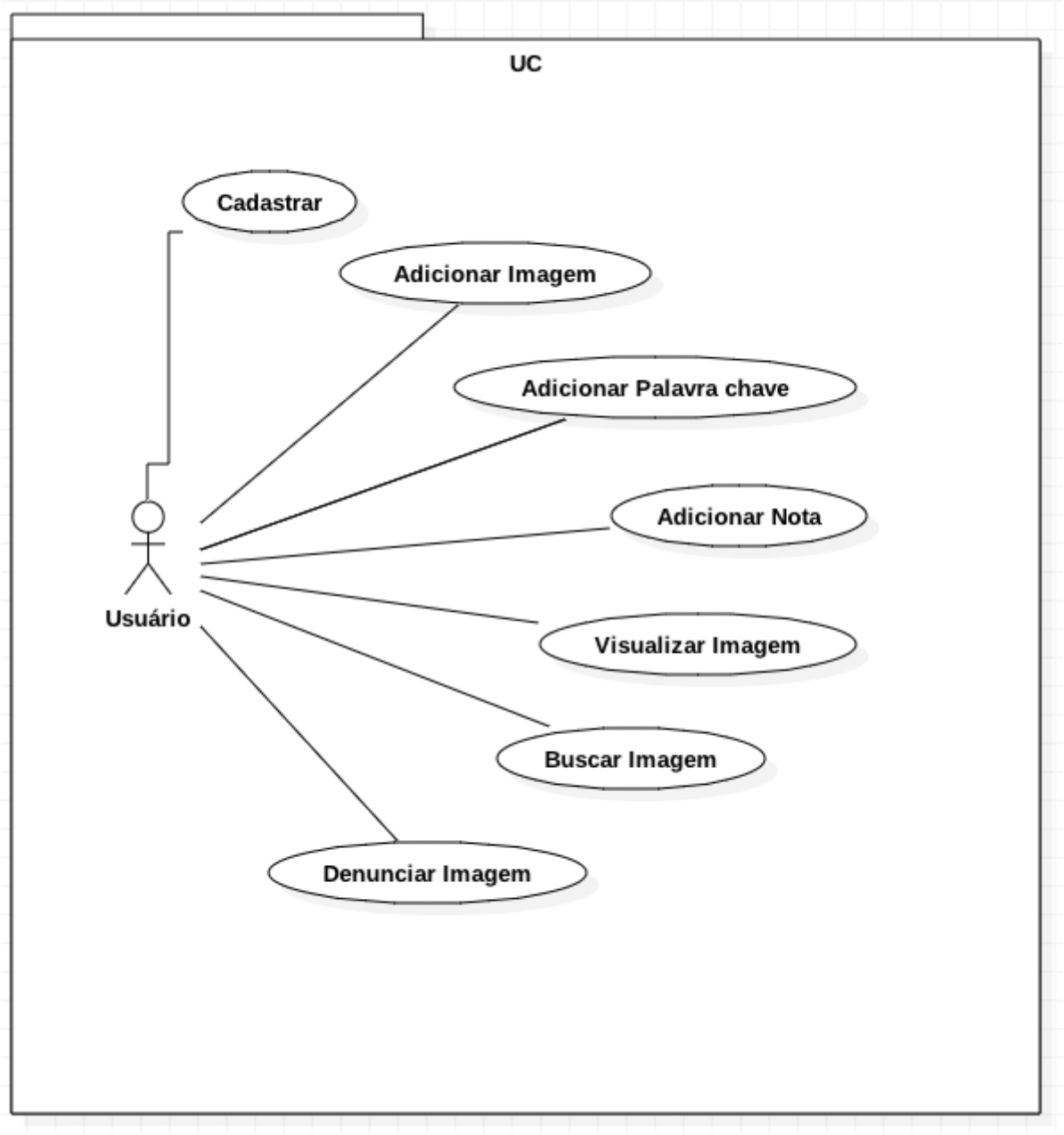

Figura 21: Diagrama de Casos de Uso.

A.2.2 UC1 - CADASTRO DE USUÁRIOS

- **Ator principal:** Usuário.
- **Descrição:** Caso de uso executado para permitir o cadastro do usuário.
- **Pré-condições:** O sistema deverá estar funcionando corretamente e o usuário conectado a internet para acessar o serviço de cadastro.
- **Pós-condições:** O usuário estará cadastrado no sistema.
- **Fluxo básico:**
  1. O usuário envia as informações para o sistema
  2. O sistema responde ao usuário sucesso ou erro.

A.2.3 UC2 - AUTENTICAÇÃO DE USUÁRIO

- **Ator principal:** Usuário.
- **Descrição: Caso de uso executado para permitir que o usuário se autentique junto ao sistema.**
- **Pré-condições:**
  - Usuário deverá estar cadastrado no sistema e autenticado.
- **Pós-condições:** O usuário estará autenticado no sistema.
- **Fluxo básico:**
  1. O usuário envia suas credenciais para o sistema
  2. O serviço responde com o perfil do usuário e um token de acesso ao sistema.
- **Fluxo Alternativo:**
  1. O usuário envia suas credenciais para o sistema
  2. O serviço responde com uma mensagem de erro informando que não foi possível autenticar o usuário.

A.2.4 UC3 - ADIÇÃO DE IMAGEM

- **Ator principal:** Usuário.
- **Descrição: Caso de uso executado para permitir que o usuário envie uma imagem que esteja geolocalizada para o sistema.**
- **Pré-condições:**
  - –Usuário deverá estar cadastrado no sistema e autenticado.
  - –A imagem que o mesmo deseja enviar deverá ser geolocalizada através dos dados EXIF.
- **Pós-condições:** A imagem enviada foi salva no servidor com sucesso.
- **Fluxo básico:**
  1. O usuário se autentica no sistema.
  2. Seleciona a imagem que deseja enviar.
  3. Envia a imagem para o serviço.
  4. O serviço responde se houve sucesso ou falha no envio da imagem.

A.2.5 UC4 - ADIÇÃO DE PALAVRAS-CHAVE

- **Ator principal:** Usuário.
- **Descrição:** Caso de uso executado para permitir que o usuário adicione palavras-chave (*tags*) a imagens.
- **Pré-condições:**
  - –O usuário necessita estar autenticado no sistema.
  - –O usuário necessita selecionar a imagem ao qual deseja adicionar as palavras-chave.
- **Pós-condições:** Palavras-chave adicionadas a imagem serão exibidas a todos que selecionarem a mesma para visualização e também aparecerão no mecanismo de busca.
- **Fluxo básico:**

1.O usuário se autentica no sistema.

2.O usuário escolhe uma imagem que deseje adicionar palavras-chave (*tags*).

3.O usuário escreve as mesmas em um local apropriado.

4.O usuário envia essas novas palavras-chave para o serviço armazenar as alterações.

A.2.6 UC5 - ADIÇÃO DE NOTA

•**Ator principal:** Usuário.

•**Descrição:** Caso de uso executado para permitir que o usuário possa adicionar notas nas imagens que ele enviou para o serviço.

•**Pré-condições:**

–O usuário deverá estar autenticado.

–O usuário devera selecionar a imagem que deseja adicionar uma nota.

•**Pós-condições:** A nota que o usuário escreveu estará disponível para quem selecionar a imagem possa ler.

•**Fluxo básico:**

1.O usuário se autentica no sistema.

2.O usuário escolhe uma imagem que seja de autoria própria para adicionar a nota.

3.O usuário escreve a nota.

4.O usuário envia essa nota para o serviço armazenar as alterações.

A.2.7 UC6 - VISUALIZAR IMAGENS

•**Ator principal:** Usuário.

•**Descrição:** Caso de uso executado para permitir que o usuário visualize as imagens.

•**Pré-condições:** O usuário deverá estar conectado a internet e estar com o aplicativo aberto no dispositivo.

•**Pós-condições:** Um mapa com imagens aparecerá na tela ou uma imagem caso o usuário tenha selecionado alguma do mapa.

•**Fluxo básico:**

1.O usuário inicia o aplicativo.

2.O usuário escolhe uma foto caso deseje ver a mesma ampliada.

A.2.8 UC7 - BUSCA DE IMAGENS

•**Ator principal:** Usuário

•**Descrição:** Caso de uso executado para permitir que o usuário faça uma busca por imagens.

•**Pré-condições:** O usuário precisará estar conectado à internet e com o aplicativo aberto.

•**Pós-condições:** Aparecerá uma lista de imagens para o usuário.

•**Fluxo básico:**

1.O usuário adicionará as palavras-chave pelas quais deseja buscar as imagens.

2.O usuário tocará no botão buscar.

3.O usuário poderá selecionar uma imagem das retornadas em uma lista para ampliar.

A.2.9 UC8 - DENÚNCIA DE IMAGENS

•**Ator principal:** Usuário.

•**Descrição:** Caso de uso executado para permitir que o usuário possa denunciar uma imagem.

•**Pré-condições:**

–O usuário precisará estar autenticado no sistema.

–O usuário precisará ter selecionado a imagem que achar imprópria.

–O usuário precisará adicionar um motivo pelo qual acha que a imagem não deveria estar sendo exibida.

•**Pós-condições:** A denúncia é enviada ao serviço.

•**Fluxo básico:**

1.O usuário se autentica no sistema.

2.O usuário escolhe a imagem que acha imprópria.

3.O usuário preenche o formulário adicionando o motívo pelo qual o mesmo acha que a imagem é indevida.

4.O usuário envia a denúncia ao serviço.