Coleta e Persistência de Informação Geográfica Voluntária

Vanessa Souza*
UNIFEI - Universidade Federal de Itajubá
Itajubá, Minas Gerais - Brasil
vanessasouza@unifei.edu.br

Eduardo Faggiani
UNIFEI - Universidade Federal de Itajubá
Itajubá, Minas Gerais - Brasil
eduggiani@live.com

ABSTRACT

Volunteered geographic information - VGI has gained visibility due to the variety of applications that can benefit from its use, including environmental monitoring, tourism, security, biodiversity, agriculture, among others. Although widespread, there is a lack in the literature on technical details of mobile VGI applications. The objective of this work was to demonstrate the development of a mobile VGI application, focusing on collection and persistence of this data type, including digital media. For this, an APP was developed. MySQL Spatial and MongoDB databases were evaluated for read and write latency. The work details the development of the APP, listing the requirements, justifying the choices and illustrating code; and presents the results of the performance tests, demonstrating that for small VGI applications, MySQL Spatial can be a good alternative as a persistence mechanism.

KEYWORDS

Informação Geográfica Voluntária, Persistência de Imagem, Mobile

1 INTRODUÇÃO

A informação geográfica se caracteriza por possuir uma forma de localização na superfície da Terra associada por coordenadas geográficas (latitude e longitude), ou outras formas de localização, como endereço. Por sua vez, a Informação Geográfica Voluntária, do inglês *Volunteered Geographic Information (VGI)*, é conceituada como o engajamento de cidadãos na criação de informação geográfica (ou geoespaciais) de maneira voluntária e sem treinamento específico [10]. Sendo assim, a VGI se torna útil no aprimoramento dos processos de tomada de decisão por parte dos gestores, pois, para alguns autores [10, 27], os seres humanos podem ser comparados a "sensores móveis inteligentes", uma vez que são capazes de adquirir informações geoespaciais acuradas por conhecerem o lugar que vivem e terem sensibilidade suficiente para ponderar sobre as situações.

Vale ressaltar que o avanço na computação móvel — incluindo *hardware*, redes de telecomunicação e ainda a evolução de aplicativos que lidam com conteúdo geoespacial — tem transformado a forma como esses dados são coletados, armazenados, disponibilizados, analisados, visualizados e utilizados [6]. Grande parte das aplicações VGI apresenta um componente *mobile*, pois o dispositivo móvel possui todos os instrumentos necessários para uma coleta de informação geoespacial fácil, intuitiva e democrática. Com *smartphones*, é possível coletar informações em formato de texto, imagem, som ou vídeo, sendo que o GPS integrado aos aparelhos é utilizado para georreferenciar o dado coletado. Estima-se que 3.5 bilhões de pessoas no mundo utilizam dispositivo móvel no seu dia-a-dia [1], deixando claro que os *smartphones* impactam o estilo de vida das pessoas e alteram a maneira como elas trabalham, fazem compras, viajam e se comunicam. Assim, a VGI reflete essa mudança, tanto tecnológica, quanto no estilo de vida dos cidadãos.

No entanto, a evolução de aplicações VGI fez surgir demandas tecnológicas, e, atualmente, não há padronização para implementação de projetos VGI, especialmente no que tange o aspecto de persistência dos dados.

Nesse contexto, esse trabalho pretende demonstrar o desenvolvimento de uma aplicação *mobile* VGI, focando na coleta e persistência desse tipo de dado. Para tanto, um aplicativo para dispositivos móveis foi desenvolvido, e dois mecanismos de persistência de dados — sendo um relacional e um NoSQL — foram utilizados no *back-end* da aplicação, para avaliar desempenho. Destaca-se que a literatura apresenta algumas aplicações, mas há uma carência no detalhamento dos aspectos tecnológicos envolvidos no desenvolvimento. Especialmente quando se fala de aplicações VGI menores, ou seja, aquelas que pretendem trabalhar com nichos específicos.

A contribuição do estudo está em detalhar as etapas de desenvolvimento da aplicação VGI, apresentando as soluções utilizadas e facilitar o desenvolvimento de novas aplicações.

O restante deste artigo está dividido em cinco seções, a saber: a seção 2 apresenta a revisão bibliográfica; a seção 3 apresenta metodologia. A sessão 4 apresenta o desenvolvimento do aplicativo *mobile* VGI; na sessão 5 são apresentados os resultados dos testes de desempenho. Na seção 6 apresenta-se as considerações finais.

2 INFORMAÇÃO GEOGRÁFICA VOLUNTÁRIA

Segundo Longley e colaboradores [19], o adjetivo 'geográfico' associado à um dado e/ou informação se refere à superfície da Terra e ao que está próximo da superfície. Os autores esclarecem ainda que o termo 'espacial' — comumente utilizado para se referir a um dado/informação geográfica — pode ser utilizado em qualquer espaço, não apenas referindo-se à superfície da Terra. Por fim, existe ainda o termo 'geoespacial', que, segundo os autores, refere-se ao subconjunto do termo 'espacial' aplicado à superfície da Terra. O fato é que todos esses termos são utilizados para se referir a um tipo de informação que carrega consigo um aspecto geográfico. Ou seja, possuem informações como coordenada geográfica (latitude/-longitude), endereço, CEP, cidade, entre outros. Nesse contexto, os dados geoespaciais se diferem dos convencionais pelo fato de conterem uma componente espacial e buscarem ser representações da superfície terrestre, relacionando-se ao seu posicionamento no globo.

Retratando muito adequadamente o uso da informação geográfica, Turner [28] definiu o termo *Neogeography*, ou 'Nova Geografia'.

*Instituto de Matemática e Computação - IMC

[1] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

Segundo o autor, *nova geografia é sobre pessoas que usam e criam seus próprios mapas, em seus próprios termos e combinam elementos de um conjunto de ferramentas existente. A nova geografia é sobre o compartilhamento de informações de localização com amigos e visitantes, ajudando a moldar o contexto e transmitindo a compreensão através do conhecimento do local (TURNER, 2006, p. 3).*

Nesse contexto, dados geoespaciais são criados diariamente por meio de atividades cotidianas, por cidadãos utilizando seus *smartphones* com sensores GPS ativos, ou por meio de ferramentas como *Google Maps* [2], Wikimapia [3] e *Open Street Maps* [4], que possibilitam ao usuário criar seu próprio mapa *online* ou contribuir com mapas já criados, dentro do conceito de mapeamento colaborativo. Os indivíduos, que antes eram agentes passivos, hoje agem como produtores e consumidores desses dados. Tal engajamento fez surgir uma nova demanda tecnológica, capaz de suportar a geração, armazenamento e distribuição dessas informações, que podem pertencer a qualquer tipo de mídia: texto, imagem, som ou vídeo [25].

A informação geográfica gerada por pessoas comuns — utilizando tecnologias como a web e aplicativos móveis — foi denominada de Informação Geográfica Voluntária (do inglês *Volunteered geographic information - VGI* [10]. A VGI difere da informação geográfica tradicional (ou, profissional), por ser gerada por voluntários sem treinamento. Esse tipo de informação tem hoje uma grande importância, uma vez que tem sido utilizada como origem e/ou enriquecimento de bases de dados que dão suporte a uma ampla gama de serviços, em diferentes áreas, como monitoramento ambiental, relatórios de eventos, análise de movimento humano, gestão de desastres, turismo, segurança, biodiversidade, agricultura, dentre várias outras áreas que podem se beneficiar do uso de informações adquiridas com VGI [25].

Uma característica importante da VGI é se ela está sendo gerada de forma implícita ou explícita [1]. A VGI implícita é aquela em que o usuário insere a informação geoespacial a partir de uma aplicação que não tem esse foco — como adicionar o local em *post* do Instagram. Já a VGI explícita é oriunda de aplicações nas quais a geografia e o aspecto espacial da informação são colocadas em primeiro lugar. — como a inclusão de um ponto de interesse no *Google Maps*. Outra questão é se a informação é gerada de forma ativa ou passiva, sendo VGI ativa aquela em que o usuário executa a ação de obter a informação, e a VGI passiva aquela obtida apenas por sensores, como o GPS do celular.

Mais recentemente, a VGI foi incluída em projetos de ciência cidadã (*citizen science*) [12], que, apesar de terem finalidades científicas, esses projetos usam não cientistas como voluntários [13]. Projetos científicos baseados em ciência cidadã ganharam notoriedade nas últimas décadas e são classificados em: computação voluntária, pensamento voluntário e sensoriamento voluntário. A VGI está intimamente relacionada ao sensoriamento voluntário, no qual os cidadãos colaboram com a captação de dados úteis em investigações científicas. A integração da VGI com o sensoriamento voluntário amplia o conceito de VGI porque o usuário passa a ter uma tarefa específica para ser realizada, que pode ser executada de forma ativa ou passiva. Mas, normalmente, configura-se como uma VGI explícita.

[2]https://www.google.com.br/maps
[3]https://satellite-map.gosur.com/
[4]https://www.openstreetmap.org/

Diversas aplicações VGI são implementadas por meio de aplicações *mobile* (APPs) [3, 5, 6], porque, com os *smartphones* atuais, é possível captar as diversas mídias (texto, imagem, som e vídeo), associá-las a uma localização geográfica com o auxílio do sensor GPS e beneficiar-se do avanço das redes móveis para o compartilhamento de tais informações. Atualmente, o *smartphone* é mais democrático do que os computadores pessoais, porque tem um custo menor e inclui diferentes funcionalidades. Além disso, a população está muito habituada a interagir com APPs. No entanto, o motivo mais relevante no uso de *smartphones* na coleta de dados VGI está na própria mobilidade, ou seja, em permitir que o cidadão se desloque dinamicamente para que essas informações sejam adquiridas.

Maiores detalhes sobre VGI podem ser vistos em [14, 17]. A próxima subseção trata da interação entre VGI e computação móvel.

2.1 VGI e Computação Móvel

A computação móvel é definida como o uso de dispositivos de computação portáteis com tecnologias de comunicação móvel [16]. Nesse paradigma computacional, os usuários têm acesso a serviços independentemente de sua localização, podendo inclusive, estar em movimento (mobilidade) [8].

O contraponto da mobilidade é a menor capacidade computacional dos dispositivos móveis [5] — incluindo processamento, armazenamento e energia [8]. Ou seja, os *smartphones* são ferramentas apropriadas para a aquisição de VGI — ainda que tal processo padeça com as limitações de capacidade computacional — e um mecanismo adequado para a coleta de dados pode aumentar a precisão dos mesmos e motivar o entusiasmo das pessoas [27].

O desenvolvimento de aplicações móveis deve levar em consideração a interface com o usuário (*front-end*); e a parte do servidor (*back-end*). Em função de sua arquitetura, a quantidade de requisições feitas ao *back-end* por dispositivos móveis é comumente maior do que as feitas por um dispositivo fixo. Sendo assim, o servidor deve ser otimizado para suportar um número maior de requisições. No que se diz respeito ao *front-end*, este possui um grande impacto na questão de atrair voluntários e promover os conceitos acerca de VGI ao público — em especial, engajamento. Por isso, recomenda-se que seja amigável, e que o voluntário tenha uma experiência agradável na utilização.

As aplicações para dispositivos móveis são classificadas em Nativas, Web e Híbridas [21]:

- **Nativos :** A programação é feita com base no sistema operacional no qual será executado (Android, IOS, etc.). A aplicação desenvolvida pode ser executada sem a necessidade de internet (*offline*). Normalmente, apresenta maior desempenho, porém a atualização do aplicativo é dificultada.
- **Web :** São basicamente sites para dispositivos móveis, que dependem de um navegador e conexão com a internet. A interface não possui os elementos nativos do aparelho.
- **Híbridos :** Combinam os elementos dos aplicativos nativos com o web. Utilizam o navegador, mas conseguem acessar os recursos do dispositivo. As principais desvantagens associadas a esse tipo de aplicação são desempenho, segurança e impossibilidade de execução *offline*.

[5]Nesse trabalho, focaremos o conceito de dispositivos móveis nos telefones celulares que acessam à internet, ou seja, os *smartphones*.

Em relação à VGI, considera-se que a execução *offline* seja uma característica preponderante, pois permitirá um maior engajamento. Isso porque o fato de o usuário poder acessar o aplicativo de onde estiver, sem a necessidade da conexão síncrona ao uso do APP, democratiza e melhora a experiência de uso, e, com isso, a possibilidade da coleta da informação aumenta. Nesse sentido, as aplicações nativas são mais apropriadas para a VGI do que as Web e Híbridas.

Além de funcionar *offline*, outras características são apontadas como importantes em uma aplicação *mobile* pra VGI [18] : interface simples e amigável, anonimato e facilidade para *download*.

2.2 Persistência de Dados Geoespaciais

Como dito, os dados geoespaciais têm como característica a informação geográfica associada. Ou seja, o dado deve ser armazenado considerando sua localização geográfica (latitude e longitude). Os Sistemas Gerenciadores de Banco de Dados (SGBDs) espaciais são mecanismos de persistência capazes de gerenciar dados com representação geométrica. Isso significa que eles possuem tipos de dados que a) representam a geometria (ponto, linha, polígono e/ou um conjunto destes); b) estendem a linguagem de consulta para trabalhar sobre essas geometrias implementando os operadores espaciais e topológicos; e c) permitem otimizações de armazenamento e acesso, como os índices espaciais [4].

Nos SGBDs relacionais, a maioria trabalha com as chamadas extensões espaciais, desenvolvidas sobre um SGBD Objeto-Relacional [6]. Pode-se citar como exemplo o Oracle Spatial, PostGIS e o MySQL Spatial. Já alguns SGBDs NoSQL foram implementados com funções espaciais nativas. Exemplo é o SGBD orientado da documentos MongoDB. Estudos comparando SGBDs espaciais podem ser vistos em [15, 30]

2.3 Trabalhos Correlatos

Nessa seção serão apresentados alguns estudos envolvendo coleta de VGI em aplicações *mobile*. O objetivo dessa análise foi a identificação das ferramentas (APIs, linguagens, SGBDs) utilizadas na implementação dos projetos.

O *ClickOnMap*[7] é uma plataforma web que tem como objetivo reduzir o tempo e a complexidade do desenvolvimento de múltiplos sistemas que coletam e distribuem VGI [3]. A ideia é que o interessado utilize a plataforma para criar o seu sistema de coleta e administração de dados VGI. Para tanto, a plataforma oferece uma interface administrativa e um aplicativo móvel que possibilita coleta de VGI para as aplicações diversas. O aplicativo é nativo e, portanto, permite a coleta de dados *offline*. Os autores não detalham a persistência dos dados geoespaciais.

Cruz e colaboradores [5] desenvolveram uma plataforma VGI para auxiliar na locomoção de deficientes visuais. A partir de um *smartphone*, voluntários mapeiam obstáculos que possam oferecer perigo aos cegos e pontos de interesse. O APP associa o obstáculo ao local e avisa o deficiente visual de possíveis obstáculos e sobre a presença de pontos de interesse ao seu redor. De forma análoga ao trabalho anterior, não há detalhamento técnico sobre o desenvolvimento do APP e sobre a persistência dos dados.

O Strepitus é um aplicativo VGI criado para coletar reclamações e denúncias sobre poluição sonora e ruído urbano usando dispositivos móveis ou uma página na web [6]. Na coleta a partir do dispositivo móvel (iOS e Android), o nível de ruído é medido usando o microfone presente no dispositivo utilizado. Os autores relatam que a persistência dos dados é feita utilizando o SGBD PostgreSQL/PostGIS.

Em relação à persistência de dados oriundos de aplicações VGI, o trabalho correlato com mais aderência à esse trabalho, é o de Maia e colaboradores [20]. Os autores compararam a desempenho do SGBD PostGIS com o MongoDB a partir de uma base VGI. Nos testes realizados, o MongoDB apresentou desempenho melhor que o PostGIS na maioria dos testes. O trabalho não avaliou mídias digitais, apenas texto.

Em [11], os autores apresentam um estado da arte sobre o armazenamento de dados geoespaciais em bancos NoSQL, e concluem que bancos de dados NoSQL orientados a documentos podem ser mais adequados para processamento de dados geoespaciais massivos do que os demais modelos.

Apesar da infinidade de projetos utilizando VGI encontrados na literatura, pouco se fala sobre a arquitetura tecnológica por trás desses sistemas — em especial as aplicações *mobile*. Essa percepção é reforçada em [2], que afirmam que existem poucas oportunidades de publicação sobre o gerenciamento de informações geoespaciais de *crowdsourcing* no contexto de sistemas de informações móveis. A usabilidade dessas aplicações é discutida em [26]. As premissas sobre VGI e tecnologias móveis são relacionadas em [27].

Sendo assim, o objetivo desse trabalho é apresentar um relato de experiência no desenvolvimento de um APP com a finalidade de coletar VGI, simulando um projeto de ciência cidadã. A metodologia utilizada é apresentada na próxima seção.

3 METODOLOGIA

As etapas executadas nesse projeto são apresentadas na Figura 1.

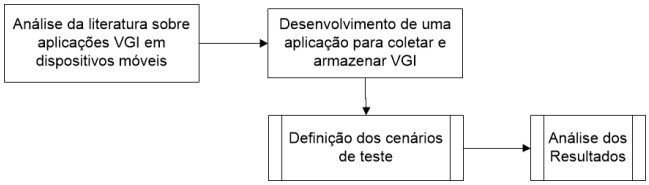

Figura 1: Fluxograma de atividades executado na metodologia.

A etapa de análise da literatura resultou nas seção 2 deste artigo. O objetivo era não só avaliar as últimas referências sobre o assunto, mas também avaliar a questão tecnológica. Avaliar quais *frameworks*, APIs, bancos de dados e linguagens de programação mais utilizadas em aplicações VGI *mobile*. Como resultado dessa etapa, verificou-se que a questão é pouco abordada na literatura.

Alguns trabalhos propõem ferramentas bastante robustas e, por isso, utilizam computação em nuvem. Em [18] há um relato mais detalhado da arquitetura utilizada em uma aplicação VGI para desastres naturais. Nesse trabalho, a persistência dos dados é feita de forma customizada, em memória. Ainda sobre a persistência, o SGBD PostGis é bastante utilizado no *back-end* [6, 29].

Portanto, considerando a falta de detalhamento desse tipo de aplicação, nesse estudo, optou-se por implementar um aplicativo

[6] SGBDs relacionais que possuem recursos para definição, manipulação e armazenamento de dados complexos, utilizando princípios da Orientação Objeto.
[7] http://www.dpi.ufv.br/projetos/clickonmap/

mobile (Android) para coleta e armazenamento de informação geográfica voluntária. O aplicativo deveria coletar informações textuais, imagens e a localização geográfica. O desenvolvimento foi dividido, portanto, em duas fases : coleta e armazenamento. A coleta é feita a partir do dispositivo móvel e os dados devem ser transferidos para o servidor *back-end*, responsável pelo armazenamento centralizado.

O APP foi desenvolvido como uma aplicação Android escrita na linguagem de programação Java e utilizando a IDE Android Studio (versão 3.2.1). O Android é um sistema operacional para dispositivos móveis, baseado em Linux e desenvolvido pela empresa Google e pelo consórcio *OHA's Open Handset Alliance*.

O Java é a linguagem oficial do Android, possibilita melhor desempenho para o aplicativo e possui muitas bibliotecas nativas à disposição do desenvolvedor. Apesar de atualmente existir diferentes linguagens (Kotlin) e *frameworks* (Flutter, React Native, Ionic) para desenvolvimento *mobile*, optou-se por utilizar a linguagem oficial nesse trabalho. Destaca-se aqui que, no ano de 2019, a empresa Google anunciou que a linguagem Kotlin passaria a ser a oficial para desenvolvimento Android. No entanto, em comparações envolvendo as duas linguagens, o Java obteve melhor desempenho [9, 24]. Além disso, nos rankings atuais[8] Java ainda é a linguagem mais utilizada.

A persistência local dos dados foi feita utilizando o SGBD relacioanl SQLite [9], projetado para dispositivos com recursos de *hardware* escassos. Por isso, apresenta algumas limitações como nos tipos de dados suportados (text, integer e real). O SQLite foi escolhido nesse trabalho porque o Android garante suporte nativo a esse SGBD. Portanto, no dispositivo móvel, os dados coletados são armazenados no SQLite e, como será explicado na Seção 4, a imagem é armazenada em formato BASE64 [10].

O *back-end* foi implementado em PHP. Foram avaliados quatro cenários distintos, combinando o mecanismo de persistência e a forma de armazenamento da imagem. Em relação aos mecanismos de persistência, avaliou-se o uso do SGBD Relacional Mysql e do SGBD NoSQL orientado a documentos MongoDB. O MySQL fornece uma extensão que garante suporte à geração, análise e armazenamento dados geoespaciais (*Mysql Spatial*) e é aderente ao OpenGIS [11]. Ele possui tipos de dados para representação espacial (ponto, linha, polígono), funções para manipulação desses dados e indexação espacial. O MongoDB oferece suporte nativo a dados geoespaciais. Já com relação a forma de armazenamento da imagem, duas abordagens foram utilizadas: armazenar a imagem no banco de dados, no formato BASE64 e armazenar a imagem em um servidor de arquivos. Nesse caso, o banco armazena apenas o endereço da imagem. Portanto, os quatro cenários testados foram :

- **Cenário 1 :** A mídia é enviada de maneira codificada (Base64) e armazenada na forma de uma String no SGDB MySQL.
- **Cenário 2 :** A mídia é enviada de maneira codificada (Base64) e armazenada na forma de uma string no SGDB MongoDB.
- **Cenário 3 :** A mídia é decodificada e salva em um servidor de arquivos no formato PNG e sua URL salva no MySQL.
- **Cenário 4 :** A mídia é decodificada e salva em um servidor de arquivos no formato PNG e sua URL salva no MongoDB.

Para cada cenário, testes de carga e consulta foram executados utilizando o *software* JMeter. O JMeter (versão 5.3) é uma ferramenta gratuita que automatiza testes de estresse e carga em banco de dados e aplicações Web, e, neste trabalho, foi utilizado para medir o desempenho (latência) dos dois sistemas de banco de dados (MySQL e MongoDB), nos cenários acima mencionados. O objetivo principal era saber se o banco NoSQL poderia ter desempenho melhor em cenários nos quais se sabe que o Relacional apresenta deficiências, como no armazenamento de mídias, em função da escalabilidade. Avaliou-se também se o fato do MongoDB ser nativamente geoespacial traria algum impacto na carga e consulta das informações.

4 DESENVOLVIMENTO DO APP VGI

O cenário pensado para o APP foi sobre a turbidez da água [12] de um lago. O voluntário deverá utilizar o APP para informar sua percepção sobre a turbidez, dando uma nota de 0 a 4 — sendo 0 uma água límpida e transparente, e a nota 4 uma água com turbidez alta. Além disso, o voluntário deverá tirar uma fotografia e, caso queira, pode deixar comentários.

Além das informações que devem ser coletadas pelo voluntário, alguns requisitos foram definidos para criação do APP:

- **[REQ01] O aplicativo deve trabalhar de forma *offline* (sem dependência de internet) :** Para atender esse requisito, o APP desenvolvido será nativo (Android) e precisará armazenar os dados localmente no dispositivo.
- **[REQ02] O dado coletado deve ser geoespacial :** O aplicativo deve coletar coordenadas geográficas (latitude e longitude) obtidas através do GPS do aparelho móvel. Essa informação é transparente para o usuário.
- **[REQ03] O dado coletado deve conter informação temporal :** O aplicativo deve armazenar o dia e horário em que o dado foi coletado. Essa informação é transparente para o usuário.
- **[REQ04] O aplicativo deve, em momento oportuno, sincronizar as informações locais com o servidor de banco de dados :** Nesta fase os dados armazenados internamente no *smarthphone* devem ser enviados para o servidor de banco de dados do projeto. Após a sincronização, os dados são removidos do aparelho móvel do usuário.
- **[REQ05] Anonimato :** Não é solicitado ao usuário que faça *login* no sistema.

A tela principal do aplicativo pode ser vista na Figura 2. Ao carregar o APP, o usuário é convidado a tirar a foto do lago. Após selecionar a imagem (Figura 2d), a mesma é carregada para o APP (Figura 2a). O usuário então dá uma nota para a turbidez (Figura 2b) e, caso queira, pode adicionar um comentário (Figura 2c). Ao finalizar a coleta, ele deve escolher a opção "Enviar"(Figura 2e). Em

[8] Dez melhores linguagens de programação para APP em 2020 - https://medium.com/front-end-weekly/10-best-mobile-development-programming-languages-in-2020-77439f9b10c1

[9] https://www.sqlite.org/

[10] Método para codificação de dados frequentemente utilizado para transmitir dados binários por meios de transmissão que lidam apenas com texto

[11] Consórcio que cria padrões para definição e compartilhamento de dados geoespaciais. https://www.ogc.org/

[12] A medida da dificuldade de um feixe de luz atravessar certa quantidade de água é chamada de Turbidez — quanto maior a turbidez, menos luz atravessa a camada de água. Esse é um índice importante para avaliar a potabilidade da água e, normalmente, é medido com o auxílio de sensores.

momento oportuno, ele deverá sincronizar os dados com o servidor central (Figura 2f).

É importante ressaltar que o aplicativo desenvolvido nesse trabalho não tem a intenção de ser um produto para o usuário final, mas sim um ambiente de testes para tecnologias VGI. Portanto, algumas funcionalidades básicas de aplicações VGI não foram implementadas, tais como visualização geoespacial dos dados e informações estatísticas sobre o alvo da aplicação, neste caso, a turbidez.

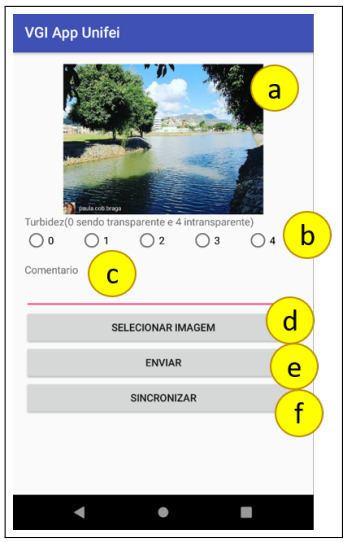

Figura 2: Tela principal do aplicativo *mobile* e suas principais funcionalidades.

Atendendo ao requisito de trabalhar *offline* (REQ01), uma tabela é criada no dispositivo do voluntário. O trecho de código 1 detalha essa implementação. Esse código é chamado apenas na primeira vez que o aplicativo é executado no dispositivo. Entre as linhas 1 e 9, o código recebe as variáveis. Entre as linhas 10 e 12, a tabela é criada no SQLite.

```
1  public static final String TABLE_NAME = "VGI";
2  public static final String COL1 = "ID";
3  public static final String COL2 = "Comentario";
4  public static final String COL3 = "turbidez";
5  public static final String COL4 = "latitude";
6  public static final String COL5 = "longitude";
7  public static final String COL6 = "foto";
8  public static final String COL7 = "status";
9  public static final String COL8 = "data";.
10 public void onCreate(SQLiteDatabase db) {
11 String createTable = "CREATE TABLE " + TABLE_NAME + "(" + COL1 + " INTEGER
       PRIMARY KEY AUTOINCREMENT, " + COL2 + " TEXT, " + COL3 + " INTEGER, " +
       COL4 + " REAL, " + COL5 + " REAL, " + COL6 + " TEXT, " + COL7 + " INTEGER
       , " + COL8 + " TEXT)";
12    db.execSQL(createTable);}
```

Trecho de Código 1: criação da tabela que armazena os dados localmente no dispositivo móvel utilizando o SQLite.

Após a criação da tabela, o voluntário (usuário) pode enviar os dados previamente coletados (Figura 2e). O atributo **ID** (linha 2 do trecho de código 1) é gerado automaticamente como um número inteiro *autoincrement*. Ele serve como chave primária da tabela. Os atributos **comentário**, **turbidez** e **foto** vêm dos dados coletados pelo usuário. As coordenadas geográficas (**latitude e longitude**), assim como a **data** são obtidas de forma automática pelo APP.

Para obter as coordenadas geográficas (REQ02), é preciso que o GPS do celular esteja habilitado. Essa tarefa é ilustrada no trecho de código 2. A linha 1 checa se o GPS está ativo. O dado adquirido refere-se à última localização registrada do GPS em um intervalo definido de tempo e espaço (a cada 6000ms ou 10 metros - linha 2). Na linha 3, o método *getLastKnownLocation* retorna um objeto que contém coordenadas de latitude e longitude. Como o SQLite não dá suporte a dados geoespaciais, a coordenada geográfica obtida do GPS é dividida em latitude e longitude e armazenada como *real*.

```
1 if (isGPSEnabled){
2     lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 6000,10,this);
3     Location loc = lm.getLastKnownLocation(LocationManager.GPS_PROVIDER);
4     return loc;
```

Trecho de Código 2: Coleta das coordenada geográficas pelo APP.

A informação de data e hora (REQ03) é coletada diretamente do sistema utilizando o método *Calendar* e então convertida para *String* e formatada como data, já que o SQLite não tem suporte para dados tipo *date*. O trecho de código JAVA é apresentado em 3.

```
1 Date currenttime = Calendar.getInstance().getTime();
2 final String hora = dateFormat.format(currenttime.getTime());
```

Trecho de Código 3: Coleta da informação temporal - data e hora.

Em relação à imagem digital, a mesma deve ser tirada pelo voluntário e selecionada dentro do APP para ser enviada. Por ser um dispositivo móvel que trabalha *offline*, guardar a referência da imagem (endereço ou caminho no diretório do celular) pode ocasionar na perda desta caso o usuário delete as fotos de seu dispositivo antes da sincronização. Além disso, mesmo que seja guardada a referência, no momento de envio essa imagem deve ser previamente tratada. Sendo assim, optou-se por armazenar a imagem em BASE64 no SQLite.

Na linguagem Java, para que a imagem seja armazenada em BASE64, primeiro ela é transformada em um Bitmap. A função é apresentada no trecho de código 4. Na linha 3, a imagem é convertida em Bitmap de acordo com o formato da imagem (PNG), a qualidade (100) e o formato de saída (ByteArray). O objeto *ByteArray* pode então ser codificado para Base64 (linha 5).

```
1 public String BitMapToString(Bitmap bitmap){
2     ByteArrayOutputStream baos = new  ByteArrayOutputStream();
3     bitmap.compress(Bitmap.CompressFormat.PNG,100, baos);
4     byte [] b = baos.toByteArray();
5     String temp = Base64.encodeToString(b, Base64.DEFAULT);
6     return temp;}
```

Trecho de Código 4: Conversão da imagem para o formato Base64.

Com os dados salvos localmente no dispositivo móvel, o voluntário pode enviá-lo ao servidor em momento oportuno, a partir de uma requisição PHP disparada pelo botão "Sincronizar"na tela do APP (REQ04).

No *Back-End* os dados são tratados para serem inseridos no banco relacional (MySQL) e o NoSQL (MongoDB). A principal atividade dessa etapa está em tratar os dados de acordo com os cenários definidos nas Seção 3. Ou seja, ora utilizar a imagem codificada em BASE64, ora decodificá-la para armazenar no servidor de arquivos. Outra atividade foi converter as informações de latitude e longitude no tipo espacial *point* dos SGBDs MySQL e MongoDB.

A Figura 3 apresenta os modelos implementados nos bancos. A Figura 3a refere-se ao modelo MySQL implementado para armazenar a imagem no banco (Base64). A Figura 3b refere-se ao modelo MySQL implementado para armazenar o endereço da imagem. A

Figura 3c exemplifica um documento do MongoDB que, independentemente da forma de armazenamento da imagem, mantém-se o mesmo.

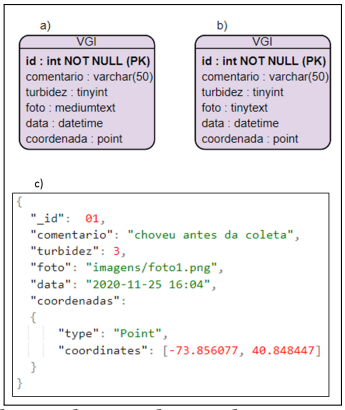

Figura 3: Modelos implementados nos bancos MySQL (a e b) e no MongoDB (c).

O requisito sobre anonimato (REQ05) é garantido pelo fato do APP não solicitar *login* e, como visto na Figura 3, não armazena informações sobre o usuário e/ou dispositivo.

5 TESTES DE DESEMPENHO

Para os testes de desempenho foram criados quatro *scripts*, sendo dois para carga e outros dois para consulta de dados. Isso porque, para cada tarefa (escrita e leitura), os cenários discutidos na Seção 3 foram avaliados variando o número de usuários. Em cada teste, a quantidade de usuários (*threads*) aumentou de 200 em 200 usuários, chegando aos limites de cada teste. Os testes foram executados no JMeter utilizando um computador com processador Intel i7, 2.3GHz, 8GB de RAM, sistema operacional 64 bits Windows. A métrica avaliada foi a latência, que mede o tempo decorrido entre o disparo e o retorno de uma requisição.

Nos teste de carga, as inserções foram feitas a partir do *script* PHP. Apesar das sintaxes de inserção nos bancos MySQL e MongoDB serem diferentes, os dados inseridos são os mesmos, mantendo assim a integridade dos testes. Os dados inseridos no banco são: Comentário, Turbidez, Imagem, Data e Coordenadas.

A Figura 4 apresenta os resultados do teste de carga em ambos os cenários. Na Figura 4a, a imagem é inserida no banco em formato BASE64. Verifica-se que a latência do MySQL é maior do que a do MongoDB. O MySQL suportou o máximo de 800 usuários, enquanto o MongoDB atingiu mil usuários. Com 800 usuários realizando requisições simultâneas, a latência do MySQL foi de 3679ms, enquanto o do MongoDB foi de 975ms. Ou seja, uma diferença de 73.49%. Já considerando a inserção do endereço da imagem no banco (Figura 4b), a latência do MySQL e do MongoDB se mantiveram parecidas até o limite próximo a mil usuários. A partir desse ponto, o MySQL estabilizou a latência em torno de 800ms, enquanto o MongoDB apresentou latências maiores. Ou seja, de forma geral, o MySQL apresentou melhor desempenho.

Comparando a Figura 4a e a Figura 4b, verifica-se que, conforme esperado, inserir apenas o endereço da imagem no banco acarreta menos processamento e, por isso, ambos SGBDs alcançaram um número maior de usuários, chegando a dois mil sem apresentar falhas.

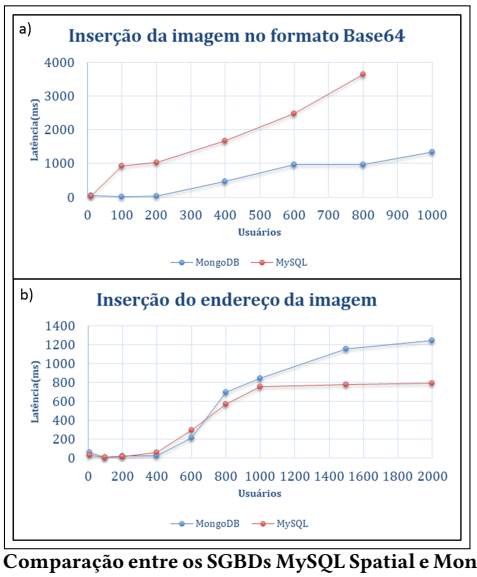

Figura 4: Comparação entre os SGBDs MySQL Spatial e MongoDB na carga de registros VGI. Em a), a imagem é armazenada como texto (BASE64); em b) apenas a URL da imagem é armazenada no banco.

Para os testes de consulta, definiu-se como critério que o teste deveria ser feito com uma consulta espacial e retornar a imagem em ambos os casos estudados (URL e Base64). A consulta definida seleciona as imagens e coordenadas em uma distância de 50 metros de um determinado ponto (-22.412142, -45.451549) e é executada via PHP. Para garantir volume suficiente para os testes, dados aleatórios foram inseridos, limitando as coordenadas em até 50 metros da localização definida. Cem mil dados foram inseridos. Mil diferentes imagens foram utilizadas aleatoriamente nesses dados.

É importante destacar que, no caso da imagem armazenada em BASE64, a latência medida considera o tempo para que a imagem seja convertida para um formato de imagem. De forma análoga, quando o banco armazena a URL da imagem, a latência medida considera a busca da imagem no diretório. O trecho de código 5 ilustra a consulta executada em PHP para quando apenas o endereço da imagem está armazenado no SGBD MySQL. O código para o SGBD MongoDB é apresentado no trecho de código 6.

```php
$sql = "SELECT foto, ST_X(coordenada), ST_Y(coordenada) FROM android.VGI WHERE
      st_distance_sphere(coordenada, POINT(-22.412142, -45.451549)) <= 50";
$result = mysqli_query($conn, $sql);
if (mysqli_num_rows($result) > 0) {
    while($row = mysqli_fetch_assoc($result)) {
        echo "Coordenadas: <br>" . $row["ST_X(coordenada)"] ."<br>" . $row["ST_Y
      (coordenada)"] ."<br>";
        echo '<img src="' . $row["foto"] . '" /> <br>';}
} else {
    echo "0 results";}
```

Trecho de Código 5: Código PHP executado nos testes de consulta com o MySQL.

```php
$query2 = array(
    'Coordenada' => array(
            '$near' => array(
                '$geometry' => array(
                    'type' => 'Point',
                    'coordinates' => array(-22.412142, -45.451549)),
                    '$maxDistance' => 50,
```

LATÊNCIA DE CARGA (ms)					LATÊNCIA DE CONSULTA (ms)				
Usuários	MySQL		MongoDB		Usuários	MySQL		MongoDB	
	Base64	URL	Base64	URL		Base64	URL	Base64	URL
10	34	31	62	61	10	14	3	26	10
100	930	12	15	12	50	13	3	25	9
200	1030	14	32	24	100	10	3	23	8
400	1673	65	477	22	200	26	3	286	13
600	2497	297	966	215	400	329	3	910	41
800	3649	573	975	697	800	1272	3	2595	683
1000	-	755	1345	844	1600	-	5	-	938
1500	-	778	-	1156	1800	-	5	-	1209
2000	-	796	-	1242					

Tabela 1: Latência medida em milissegundos nos testes de carga e consulta, com a imagem em formato Base64 e diretório (URL), nos SGBDs MySQL e MongoDB.

```
                ));
$cursor = $VGI->find($query2, array('Coordenada' => 1, 'Foto' =>1));
foreach ($cursor as $doc) {
    var_dump($doc["Coordenada"]);
    echo '<img src="' . $doc["Foto"] . '" /> <br>';  }
```

Trecho de Código 6: Código PHP executado nos testes de consulta com o MongoDB.

Os resultados dos testes de consulta são apresentados na Figura 5. Em ambos os cenários, o MySQL teve desempenho superior ao MongoDB. No entanto, no cenário em que apenas o endereço da imagem é armazenado no banco (Figura 5b), o desempenho do MySQL foi substancialmente superior ao MongoDB, especialmente a partir de 400 usuários. O gráfico deixa claro que a latência do MySQL foi constante para esse caso, enquanto a do MongoDB cresce com o aumento dos usuários. Percebe-se também que há uma diferença significativa na quantidade de usuários possíveis entre os dois cenários.

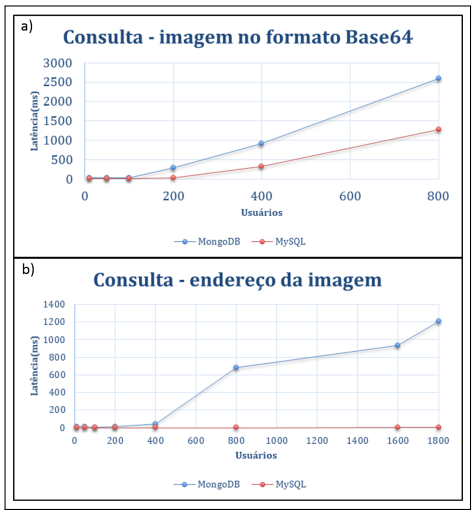

Figura 5: Comparação entre os SGBDs MySQL Spatial e MongoDB na recuperação de registros VGI. Em a), a imagem é recuperada como texto (BASE64); em b) apenas a URL da imagem é recuperada.

Os resultados numéricos dos testes apresentados nas Figuras 4 e 5 podem ser melhor analisados na Tabela 1.

No trabalho [20], os autores compararam o PostgreSQL com o MongoDB para dados oriundos de VGI. No entanto, os testes não incluíam imagens. Os resultados apresentados pelos autores mostram que, na carga, o MongoDB (servidor único) teve melhor desempenho que o PostgreSQL. No entanto, corroborando os resultados aqui apresentados, o MongoDB obteve desempenho pior que o relacional nos testes de consulta.

A latência de carga para o MySQL e o MongoDB foi avaliada em [22] fora do contexto VGI e também sem análise de mídia. Os autores também concluíram que a latência média do MySQL é sempre maior do que MongoDB para o mesmo conjunto de cargas e mesmo número de *threads*. Em relação à consulta, os autores concluem que a diferença de latência entre os SGBDs não é muito significativa e que ambos têm um desempenho quase semelhante para várias solicitações de leitura com número variável de *threads*.

Esses resultados corroboram os encontrados nos testes realizados nesse trabalho. A diferença mais significativa é a da consulta pelo endereço da imagem. Nesse caso, a diferença entre a latência do MySQL e do MongoDB foi maior.

O MySQL e o MongoDB foram avaliados quanto ao armazenamento de imagens digitais em [7, 23]. Em [7] o armazenamento da imagem em formato binário (hexadecimal) e apenas o endereço da imagem foi avaliado e os resultados corroboram os encontrados nesse trabalho. Ou seja, o MongoDB apresenta menor latência de escrita no cenário onde a imagem é armazenada no banco. No cenário em que apenas o endereço da imagem é armazenado, os autores concluíram que a latência do MySQL é menor que a do MongoDB. No entanto, os autores testaram o limite de 80 usuários [7]. Nos resultados obtidos nesse trabalho (Figura 2b), a latência de escrita do MySQL ficou melhor depois que alcançou mil usuários. Já em relação à leitura no cenário em que a imagem é inserida no banco, em [7], os autores concluíram que a latência do MongoDB foi cerca de 9 vezes menor do que o MySQL, no limite de 80 usuários. Esse resultado não condiz com os encontrados nesse trabalho. Verifica-se na Tabela 1 que, mesmo com dez usuários, a latência do MySQL foi menor que a do MongoDB. No entanto, vale destacar que o formato de armazenamento avaliado foi diferente. Aqui, a imagem foi armazenada como texto Base64; no trabalho de referência, a mesma foi armazenada em formato BLOB, como hexadecimal. Já a leitura no cenário de armazenamento apenas do endereço da imagem, os resultados de [7] corroboram os encontrados nesse trabalho. Isto é, a latência do MySQL é menor nesse caso.

6 CONSIDERAÇÕES FINAIS

Este trabalho apresentou um relato de experiência no desenvolvimento de um aplicativo *mobile* VGI, incluindo testes de desempenho no armazenamento de dados. Os requisitos definidos para o APP foram pensados sob a perspectiva de uma aplicação VGI de pequeno porte, sem a intenção de ser uma plataforma para esse fim.

A decisão de armazenar a imagem no SQLite está atrelada a garantir que a mesma não seja perdida antes da sincronização. No entanto, essa decisão pode gerar um problema secundário, pois, para recuperar os dados do SQLite no Android, é necessário utilizar um cursor (*buffer*) que apresenta uma limitação de tamanho em *megabytes*. Ou seja, o tamanho da imagem deve ser controlado. Esse fato levou à segunda decisão de solicitar ao usuário que tire a foto com o APP do *smartphone* e não de dentro do próprio APP VGI, porque, nesse caso, a resolução caía ainda mais. Assim, em detrimento de garantir que a imagem não seria apagada do dispositivo, acredita-se que o engajamento tenha sido comprometido. Isso

porque não é confortável para o usuário sair do APP, tirar a foto e depois carregá-la.

Em relação à questão de qual SGBD espacial utilizar, os resultados mostraram que — para o cenário da aplicação e considerando uma aplicação de pequeno porte — o MySQL Spatial obteve melhor desempenho na maior parte dos testes, quando comparado ao MongoDB. Apenas quando a imagem era inserida no banco em formato BASE64 o MongoDB obteve latência menor que o MySQL para todo os usuários testados. Sendo assim, avalia-se que o MySQL Spatial pode ser uma boa alternativa para aplicações VGI de pequeno porte, especialmente aquelas que envolvem consultas espaciais.

O objetivo desse trabalho era demonstrar o desenvolvimento de um aplicativo *mobile* VGI, focando na coleta e persistência de dados. Acredita-se que os resultados apresentados cumpriram com o objetivo, uma vez que a seção 4 detalhou o desenvolvimento do APP, listando os requisitos, justificando as escolhas e ilustrando trechos de código; e a seção 5 apresentou os testes de desempenho para os SGBDs espaciais avaliados.

A continuação desse trabalho será no estudo da integração dos dados oriundos de uma aplicação VGI com mecanismos que auxiliem na tomada de decisão. No caso do APP desenvolvido, estudos de mineração de dados incluindo a nota que o usuário deu para a turbidez da água, a fotografia e a localização espacial podem inferir sobre a potabilidade da água e disparar avisos para a comunidade e gestores. Esses avisos podem ser enviados pelo próprio APP. Acredita-se que com esses *feedbacks* a comunidade se interesse mais pelo problema abordado e contribua ainda mais em projetos VGI.

REFERÊNCIAS

[1] Vyron Antoniu. 2016. Volunteered geographic Information measuring quality, understanding the value. *GEOmedia* 20, 1 (2016).
[2] Gloria Bordogna and Paola Carrara. 2017. *Mobile information systems leveraging volunteered geographic information for Earth observation.* Vol. 4. Springer.
[3] J. H.S Câmara, L. F.M Vegi, R. O. Pereira, Z. A Geöcze, J. Lisboa-Filho, and W. D. de Souza. 2017. ClickOnMap: a platform for development of volunteered geographic information systems. In *12th CISTI*. IEEE, 1–6.
[4] Marco Antonio Casanova, Gilberto Câmara, Clodoveu Davis, Lúbia Vinhas, and Gilberto R Queiroz. 2005. Banco de dados geográficos. (2005).
[5] Igor GM Cruz, Cláudio EC Campelo, and Cláudio de Souza Baptista. 2017. Plataforma VGI para auxilio a navegacao de deficientes visuais.. In *GEOINFO*. 163–168.
[6] Clodoveu A Davis Jr, Hugo de Souza Vellozo, and Michele Brito Pinheiro. 2013. A Framework for Web and Mobile Volunteered Geographic Information Applications.. In *GeoInfo*. Citeseer, 147–157.
[7] Jonathan de Oliveira Assis, Vanessa CO Souza, Melise MV Paula, and João Bosco S Cunha. 2017. Performance evaluation of NoSQL data store for digital media. In *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 1–6. https://doi.org/10.23919/CISTI.2017.7975844
[8] Carlos MS Figueiredo and Eduardo Nakamura. 2003. Computação móvel: Novas oportunidades e novos desafios. *T&C Amazônia* 1, 2 (2003), 21.
[9] Stylianos Gakis and Niclas Everlönn. 2020. Java and Kotlin, a performance comparison. https://www.diva-portal.org/smash/get/diva2:1443070/FULLTEXT01.pdf
[10] Michael F Goodchild. 2007. Citizens as sensors : the world of volunteered geography. November (2007), 211–221. https://doi.org/10.1007/s10708-007-9111-y
[11] Dongming Guo and Erling Onstein. 2020. State-of-the-Art Geospatial Information Processing in NoSQL Databases. *ISPRS International Journal of Geo-Information* 9, 5 (2020), 331.
[12] Muki Haklay. 2013. Citizen science and volunteered geographic information: Overview and typology of participation. In *Crowdsourcing geographic knowledge*. Springer, 105–122.
[13] Eric Hand. 2010. People power: networks of human minds are taking citizen science to a new level. *Nature* 466, 7307 (2010), 685–688.
[14] Sara Harrison. 2020. Volunteered Geographic Information for people-centred severe weather early warning: A literature review. *Australasian journal of disaster and trauma studies* 24, 1 (2020), 3–22.
[15] Gabriel Ingemarsson. 2019. *Database Performance for GIS: A Comparison of Database Schemas for Measurements with Spatial Attributes.* Dissertação(Mestrado). KTH Royal Institute of Technology.
[16] David B Johnson and D Maltz. 1996. Mobile computing.
[17] David Jonietz, Vyron Antonio, Linda See, and Alexander Zipf. 2017. Highlighting current trends in volunteered geographic information. *International Journal of Geo-Information* (2017). https://doi.org/10.3390/ijgi6070202
[18] Linna Li and Manju Narmada Ulaganathan. 2017. Design and development of a crowdsourcing mobile app for disaster response. In *2017 25th International Conference on Geoinformatics*. IEEE, 1–4.
[19] Paul A Longley, Michael F Goodchild, David J Maguire, and David W Rhind. 2009. *Sistemas e ciência da informação geográfica.* Bookman Editora.
[20] D. C. M. Maia, B. D. C. Camargos, and M. Holanda. 2018. Performance analysis on voluntary geographic information systems with document-based NoSQL database. In *Developments and Advances in Intelligent Systems and Applications*. Springer, 181–197.
[21] Ghita K Mostefaoui and Faisal Tariq. 2018. *Mobile Apps Engineering: Design, Development, Security, and Testing.* CRC Press.
[22] Rachit Pandey. 2020. *Performance Benchmarking and Comparison of Cloud-Based Databases MongoDB (NoSQL) Vs MySQL (Relational) using YCSB.* Technical Report. https://doi.org/10.13140/RG.2.2.10789.32484
[23] D Revina Rebecca and I Elizabeth Shanthi. 2016. A NoSQL Solution to efficient storage and retrieval of Medical Images. *International Journal of Scientific & Engineering Research* 7, 2 (2016), 545–549.
[24] P. Schwermerkth. 2018. *Performance Evaluation of Kotlin and Java on Android Runtime.* Mestrado (Dissertação). Royal Institute of Technologyschool of Electrical Engineering and Computer.
[25] Hansi Senaratne, Amin Mobasheri, Ahmed Loai, and Cristina Capineri. 2017. A review of volunteered geographic information quality assessment methods. 31 (2017).
[26] Artemis Skarlatidou, Alexandra Hamilton, Michalis Vitos, and Muki Haklay. 2019. What do volunteers want from citizen science technologies? A systematic literature review and best practice guidelines. *JCOM: Journal of Science Communication* 18, 1 (2019).
[27] Weidong Song and Guibo Sun. 2010. The role of mobile volunteered geographic information in urban management. In *2010 18th International Conference on Geoinformatics*. IEEE, 1–5.
[28] Andrew Turner. 2006. *Introduction to neogeography.* "O'Reilly Media, Inc.".
[29] H. S. Vellozo, M. B. Pinheiro, and C. A D. Jr. 2013. Strepitus: um aplicativo para coleta colaborativa de dados sobre ruído urbano. In *IV Workshop Computação Apl Gestão do Meio Ambiente Recur Naturais*.
[30] Z. Zhou, B. Zhou, W. Li, B. Griglak, C. Caiseda, and Q. Huang. 2009. Evaluating query performance on object-relational spatial databases. In *2009 2nd IEEE International Conference on Computer Science and Information Technology*. IEEE, 489–492.