



Universidade de Brasília

Instituto de Ciências Exatas  
Departamento de Ciência da Computação

**Arquitetura de Armazenamento de Dados para  
Sistemas de Informação Geográfica Voluntária  
utilizando Banco de Dados NoSQL baseado em  
Documento**

Daniel Cosme Mendonça Maia

Dissertação apresentada como requisito parcial para  
conclusão do Mestrado em Informática

Orientadora

Profa. Dra. Maristela Terto de Holanda

Brasília  
2016

Ficha catalográfica elaborada automaticamente,  
com os dados fornecidos pelo(a) autor(a)

MM217a Maia, Daniel Cosme Mendonça  
Arquitetura de Armazenamento de Dados para  
Sistemas de Informação Geográfica Voluntária  
utilizando Banco de Dados NoSQL baseado em Documento  
/ Daniel Cosme Mendonça Maia; orientador Maristela  
Terto Holanda. -- Brasília, 2016.  
110 p.

Dissertação (Mestrado - Mestrado em Informática) -  
Universidade de Brasília, 2016.

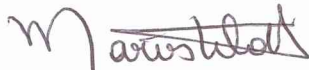
1. NoSQL. 2. Dados Geográficos. 3. Sistemas de  
Informação Geográfica Voluntária. I. Holanda,  
Maristela Terto, orient. II. Título.



**Daniel Cosme Mendonça Maia**

**Arquitetura de Armazenamento de Dados para Sistemas de Informação Geográfica  
Voluntária Utilizando Banco de Dados NoSQL baseado em Documentos.**

Tese aprovada como requisito parcial para obtenção do grau de **Mestre** no Curso de Pós-graduação em Informática da Universidade de Brasília, pela Comissão formada pelos professores:



Prof.<sup>a</sup> Dr.<sup>a</sup> Maristela Tertó de Holanda, CIC/UnB \_ Orientador



Prof. Dr. Henrique Llacer Roig, IG/UnB



Prof.<sup>a</sup> Dr.<sup>a</sup> Maria Emília Machado Telles Walter, CIC/UnB

Vista e permitida a impressão.  
Brasília, 28 de Julho de 2016.

Prof.<sup>a</sup> Dr.<sup>a</sup> Célia Ghedini Ralha  
Programa de Pós-Graduação em Informática  
Departamento de Ciência da Computação  
Universidade de Brasília

# Dedicatória

Dedico este trabalho à minha esposa Cynthia Moura, aos meus familiares, e em especial ao meu irmão gêmeo Alexandre pela força e apoio.

# Agradecimentos

Agradeço primeiramente a Deus, que me deu forças e condições para continuar essa jornada até aqui.

Agradeço a minha esposa Cynthia, pela compreensão e paciência. Havíamos acabado de nos casar quando comecei a viajar para Brasília todas as semanas para ficar praticamente todos os dias úteis fora de casa. Sei o quanto foi difícil para ela ficar a 600 km distante do marido.

Agradeço ao meu irmão gêmeo Alexandre, que me incentivou e me deu condições para permanecer em Brasília sem me preocupar com custos de moradia.

Agradeço aos meus pais (Darci e Helenice), irmãos (Katiuscia, Andréia e Gustavo), familiares mais próximos (James, Luiz Otávio, Jacira, João, Bruno e Lorena), e nossos amigos, que sempre torceram pelo nosso sucesso.

Agradeço a minha orientadora Profa. Dra. Maristela Terto de Holanda, que me proporcionou uma ótima oportunidade de trabalho e aprendizado que me deu mais tranquilidade para continuar os estudos. Além disso, agradeço pela compreensão e dedicação que apresentou durante esse período de orientação.

Agradeço aos colegas Breno Diogo e William de Branco por cederem gentilmente seus projetos para que pudéssemos realizar os testes da nossa arquitetura.

Enfim, agradeço a todos que torceram por mim!

“Se você for grato as pessoas, a sua família, a sua vida, ao seu trabalho, enfim, por tudo o que conquistou, tenho certeza que se tornará uma pessoa mais feliz!”

# Resumo

A plataforma Web 2.0 e as tecnologias móveis, como *smartphones* equipados com receptores GPS, possibilitaram uma mudança na maneira de capturar dados geográficos, contribuindo para a concepção do fenômeno dos Sistemas de Informação Geográfica Voluntária (SIGV). A partir do crescimento do número de indivíduos que criam e compartilham dados espaciais, e da possibilidade de armazenar uma grande quantidade de dados, em diversos formatos, os SIGV devem resolver algumas questões sobre como a informação pode ser armazenada e gerenciada de maneira eficiente em ambiente digital. Este trabalho visa especificar uma arquitetura de armazenamento de dados para SIGV utilizando Banco de Dados NoSQL que atenda aos requisitos de escalabilidade e heterogeneidade de dados. Para validação da arquitetura, duas provas de conceito foram apresentadas e implementadas, onde realizou-se a comparação de desempenho dos bancos de dados PostgreSQL, CouchDB e MongoDB, nas operações de inserção e leitura de dados em aplicações de SIGV. A análise das provas de conceito busca verificar a viabilidade da adoção dos bancos de dados NoSQL baseados em documento como uma alternativa para uma arquitetura de armazenamento de dados para SIGV.

**Palavras-chave:** NoSQL, Dados Geográficos, Sistemas de Informação Geográfica Voluntária.

# Abstract

The Web 2.0 platform and mobile technologies, such as smartphones equipped with GPS receivers, allowed a change in the way to capture geographic data, contributing to the design of the phenomenon of the Voluntary Geographic Information Systems (VGIS). From the growing number of individuals who create and share spatial data, and the ability to store a lot of data in various formats, VGIS should solve some questions about how the information can be stored and managed efficiently at digital environment. This work is to specify a data storage architecture for VGIS using NoSQL database that meets requirements of scalability and heterogeneity of data. For architecture validation, two proofs of concept were presented and implemented, where made the comparison of performance of PostgreSQL, CouchDB and MongoDB databases, in the operations of insertion and reading of data in VGIS applications. The analysis of the proofs of concept seeks to verify the viability of the adoption of document-based NoSQL databases as an alternative to a data storage architecture for VGIS.

**Keywords:** NoSQL, Geographic Data, Volunteered Geographic Information System.



# Sumário

Lista de Abreviaturas e Siglas	xv
<b>1 Introdução</b>	<b>1</b>
1.1 Objetivos . . . . .	3
1.2 Estrutura do Trabalho . . . . .	3
<b>2 Fundamentação Teórica</b>	<b>5</b>
2.1 Sistemas de Informação Geográfica . . . . .	5
2.2 Sistemas de Informação Geográfica Voluntária . . . . .	7
2.2.1 Wikicrimes.org . . . . .	9
2.2.2 Wikimapia.org . . . . .	10
2.2.3 OpenStreetMap.org . . . . .	11
2.3 Bancos de Dados Geográficos . . . . .	11
2.4 Sistemas de Banco de Dados NoSQL . . . . .	15
2.4.1 Escalabilidade, Replicação e <i>Sharding</i> de Bancos de Dados NoSQL . .	17
2.4.2 Tipos de Banco de Dados NoSQL . . . . .	20
2.4.3 Banco de Dados baseado em Documento ( <i>Document Store</i> ) . . . . .	22
2.4.4 Formato GeoJSON . . . . .	26
2.5 Trabalhos Relacionados . . . . .	30
2.5.1 Armazenamento de Dados Geográficos em Banco de Dados NoSQL . .	30
2.5.2 Sistemas de Informação Geográfica Voluntária . . . . .	32
<b>3 Arquitetura Proposta</b>	<b>37</b>
3.1 Descrição do Problema . . . . .	37
3.2 Proposta de Arquitetura Abstrata para SIGV . . . . .	38
3.3 Proposta de Arquitetura de Armazenamento de Dados para SIGV . . . . .	39
3.4 Implementação da Arquitetura Proposta . . . . .	44
<b>4 Provas de Conceito</b>	<b>49</b>
4.1 Prova de Conceito 1: Consulta Opinião . . . . .	49

4.2 Prova de Conceito 2: Comune . . . . .	59
4.2.1 Análise dos Testes de Inserção de Dados das Implementações da Ar- quitetura . . . . .	66
4.2.2 Análise dos Testes de Tolerância a Falhas das Implementações da Ar- quitetura . . . . .	70
4.2.3 Análise dos Resultados dos Testes das Implementações da Arquitetura	71
<b>5 Conclusão</b>	<b>74</b>
<b>Referências</b>	<b>76</b>
<b>Apêndice</b>	<b>80</b>
<b>A Estrutura dos documentos do banco de dados Comune (modelo não relacional)</b>	<b>81</b>
<b>B <i>Web services</i> implementados na arquitetura do projeto Comune</b>	<b>85</b>

# Lista de Figuras

2.1	Arquitetura de Sistemas de Informação Geográfica [12]. . . . .	7
2.2	Interface do software Wikicrimes.org. . . . .	10
2.3	Interface do software Wikimapia.org. . . . .	11
2.4	Interface do projeto OpenStreetMap.org. . . . .	12
2.5	Paradigma dos quatro universos [12]. . . . .	12
2.6	Modelo orientado a objetos básicos para dados geográficos [12]. . . . .	14
2.7	Representação vetorial de objetos geográficos e estrutura matricial de representação do espaço [12]. . . . .	15
2.8	Representação do modelo não relacional do Código 2.3 utilizando notação de Vera <i>et al</i> [60]. . . . .	25
2.9	Representação gráfica do Código 2.4: geometria ponto usando GeoJSON. . . . .	27
2.10	Representação gráfica do Código 2.5: geometria linha usando GeoJSON. . . . .	28
2.11	Representação gráfica do Código 2.6: geometria polígono usando GeoJSON. . . . .	28
2.12	Representação gráfica do Código 2.7: geometria ponto com informações adicionais usando GeoJSON. . . . .	29
3.1	Arquitetura abstrata de um SIGV utilizando Banco de Dados NoSQL. . . . .	39
3.2	Arquitetura de Armazenamento de Dados para SIGV proposta. . . . .	40
3.3	Processo de armazenamento de dados da arquitetura proposta. . . . .	43
3.4	Esquema da implementação da arquitetura utilizando um único servidor de BD. . . . .	45
3.5	Esquema da implementação da arquitetura utilizando servidor de banco de dados replicado. . . . .	46
3.6	Esquema da implementação da arquitetura utilizando <i>sharded cluster</i> . . . . .	48
4.1	Interface móvel e <i>web</i> da aplicação Consulta Opinião utilizando o MongoDB na camada de persistência de dados. . . . .	50
4.2	Modelo relacional do projeto Consulta Opinião [11]. . . . .	51
4.3	Versões do modelo não relacional do projeto Consulta Opinião. . . . .	52

4.4	Comparação dos tempos de inserção dos dados de avaliação nos bancos de dados PostgreSQL e MongoDB. . . . .	56
4.5	Comparação dos tempos de leitura dos dados de avaliação nos bancos de dados PostgreSQL e MongoDB. . . . .	57
4.6	Telas do aplicativo Comune [19]. . . . .	60
4.7	Modelo relacional resumido do banco de dados central Comune. . . . .	61
4.8	Modelo não relacional banco de dados Comune. . . . .	62
4.9	Comparativo dos tempos de inserção de dados convencionais no BD Comune.	67
4.10	Comparativo dos tempos de inserção de dados multimídia no BD Comune.	69
4.11	Comparativo dos tempos de inserção de dados geográficos no BD Comune.	69

# Lista de Tabelas

2.1 Trabalhos Relacionados: Armazenamento de dados geográficos em banco de dados NoSQL. . . . .	35
2.2 Trabalhos Relacionados: Sistemas de Informação Geográfica Voluntária. . .	36

# Lista de Códigos

2.1	Estrutura do documento JSON: Cliente (simples). . . . .	23
2.2	Estrutura do documento JSON: Cliente (dicionários). . . . .	24
2.3	Estrutura do documento JSON: Cliente, Pedido e Produto (embutidos). . .	24
2.4	Estrutura do documento GeoJSON: geometria ponto. . . . .	27
2.5	Estrutura do documento GeoJSON: geometria linha. . . . .	27
2.6	Estrutura do documento GeoJSON: geometria polígono. . . . .	27
2.7	Estrutura do documento GeoJSON: geometria ponto com informações adi- cionais. . . . .	29
3.1	Configuração do conjunto de réplicas no MongoDB replicado. . . . .	46
4.1	Estrutura do documento JSON: Usuario. . . . .	53
4.2	Estrutura do documento JSON: Questionario. . . . .	53
4.3	Estrutura do documento JSON/GeoJSON: Estabelecimento. . . . .	53
4.4	Estrutura do documento JSON: Avaliacao (Estabelecimento e Usuario em- butidos). . . . .	54
4.5	Estrutura do documento JSON: Avaliacao (Estabelecimento e Usuario refe- renciados). . . . .	54
4.6	Estrutura do documento Place do BD Comune. . . . .	62
4.7	Estrutura do documento Fs.Files do BD Comune. . . . .	63
A.1	Estrutura do documento User do BD Comune. . . . .	81
A.2	Estrutura do documento Place do BD Comune. . . . .	81
A.3	Estrutura do documento Report do BD Comune. . . . .	82
A.4	Estrutura do documento Survey do BD Comune. . . . .	83
A.5	Estrutura do documento SurveyAnswer do BD Comune. . . . .	83
A.6	Estrutura do documento Fs.Files do BD Comune. . . . .	84

# Lista de Abreviaturas e Siglas

**API** Application Programming Interface. 21, 39

**BLOB** Binary Large Object. 7

**CSV** Comma-separated values. 23

**GML** Geography Markup Language. 13

**GPS** Global Positioning System. 1

**HTML** HyperText Markup Language. 23

**HTTP** HyperText Transfer Protocol. 33, 41

**IDE** Infraestrutura de Dados Espaciais. 32, 36

**IGV** Informação Geográfica Voluntária. 1, 32

**IP** Internet Protocol. 46, 47

**ISO** International Organization for Standardization. 8

**JSON** JavaScript Object Notation. 22, 23

**MVC** Model-View-Controller. 33

**OGC** Open Geospatial Consortium. 13

**OMTG** Object Modeling Technique for Geographic Applications. 14

**PDF** Portable Document Format. 23

**REST** Representational State Transfer. 32, 41

**SGBD** Sistema Gerenciador de Banco de Dados. 6, 23, 33, 39, 43, 55

**SGBDOR** Sistema Gerenciador de Banco de Dados Objeto Relacional. 7

**SGBDR** Sistema Gerenciador de Banco de Dados Relacional. 6

**SIG** Sistemas de Informação Geográfica. 1, 5, 7

**SIGPP** Sistema de Informação Geográfica Móvel com Participação Popular. 33, 36

**SIGV** Sistemas de Informação Geográfica Voluntária. 1, 3, 5, 7, 9, 10, 31, 37–40

**SQL** Structured Query Language. 1

**SRID** Spatial Reference System Identifier. 31

**URI** Uniform Resource Identifier. 41

**WFS** Web Feature Service. 30, 35

**WMS** Web Map Service. 30, 35

**XML** eXtensible Markup Language. 22



# Capítulo 1

## Introdução

Com a facilidade e o crescimento do uso dos serviços *web* e de tecnologias móveis, como *smartphones* equipados com câmeras e dispositivos GPS (*Global Positioning System*), houve um aumento no número de indivíduos que criam e compartilham dados espaciais. A criação de informações geográficas, uma função que esteve até então reservada às agências oficiais, está sendo realizada também por cidadãos, geralmente com pouca qualificação formal na área geográfica, e de forma voluntária. A coleta de dados geográficos por usuários voluntários, trouxe impactos aos Sistemas de Informação Geográfica (SIG). Os dados coletados por esses usuários voluntários passaram a ser denominados Informação Geográfica Voluntária (IGV), e os Sistemas de Informação Geográfica que utilizam desses dados são chamados de Sistemas de Informação Geográfica Voluntária (SIGV) [26].

Um SIGV deve ter um sistema de armazenamento de dados que atenda às seguintes características: armazenamento de grande volume de dados; muitas operações concorrentes de leitura e escrita; grande número de usuários simultâneos; e a possibilidade de armazenar dados heterogêneos, estruturados ou não, a partir de diversas fontes de dados. Em face desses desafios, é necessário investigar como os dados de um SIGV podem ser armazenados, gerenciados e compartilhados, de maneira eficiente, em ambiente digital [23, 13].

Por outro lado, o sistema de Banco de Dados NoSQL (*Not Only SQL*) refere-se a um tipo de repositório de dados que não segue o tradicional modelo relacional, não possui linguagem de consulta SQL e nem esquema fixo de tabelas. Esse tipo de banco de dados é projetado para processamento distribuído, e tem a capacidade de armazenar dados em massa e realizar concorrentes operações de leitura e escrita. Os bancos de dados NoSQL têm sido cada vez mais adotados para lidar com grandes volumes de dados, e para permitir que a arquitetura de armazenamento de dados seja suficientemente flexível para possibilitar o potencial aumento do número de usuários, e da quantidade de dados armazenados na aplicação [59, 52, 46, 61, 62].

Para implementação do SIGV é comum encontrar tecnologias de armazenamento de dados que utilizam o modelo relacional. Diferentemente, a presente pesquisa propõe uma arquitetura de armazenamento de dados para Sistemas de Informação Geográfica Voluntária utilizando banco de dados NoSQL baseado em documento. Essa proposta alternativa, justifica-se pelo fato dos bancos de dados NoSQL possuírem características que vão ao encontro das necessidades dos SIGV. É importante ressaltar que esta arquitetura não somente armazena dados convencionais, mas também dados geográficos vetoriais; além de dados binários multimídia, como arquivos de imagem, áudio e vídeo, que são dados demandados em aplicações *web* e aplicativos móveis.

Como algumas das maiores preocupações dos SIGV são o armazenamento de uma grande quantidade de dados, e de dados de variados formatos, definiu-se que os requisitos de escalabilidade e heterogeneidade de dados são fundamentais para o projeto dessa arquitetura. Dessa forma, sugeriu-se a adoção da técnica de banco de dados distribuído na implementação da arquitetura, visando aumentar a capacidade de processamento e armazenamento dos dados, e ainda oferecer melhor disponibilidade e tolerância a falhas. Para lidar com os dados heterogêneos, adotou-se os bancos de dados NoSQL baseados em documento, pois oferecem flexibilidade no armazenamento de dados, não necessitando conhecer previamente a estrutura do documento para armazená-lo [13, 37, 61].

A arquitetura de armazenamento de dados proposta nesta dissertação, foi implementada utilizando três estratégias diferentes: utilizando um único servidor de banco de dados, utilizando servidores de bancos de dados replicados, e utilizando servidores de bancos de dados fragmentados. Duas provas de conceito foram apresentadas e implementadas para realizar a validação da arquitetura proposta.

A primeira prova de conceito, apresenta o projeto Consulta Opinião, que tem o objetivo de realizar pesquisas de opinião sobre estabelecimentos públicos, e apresentar os dados informados pelos usuários voluntários aos gestores desses estabelecimentos. Nessa primeira prova, comparou-se os tempos de inserção e leitura dos dados da aplicação utilizando os bancos de dados PostgreSQL e MongoDB, em um único servidor e em servidores replicados.

A segunda prova de conceito apresenta o projeto Comune, que têm objetivos semelhantes a primeira prova de conceito, mas que possui recursos adicionais como a possibilidade de trabalhar *off-line* e com dados multimídia (imagem e vídeo). Nessa segunda prova, apresentou-se os *web services* implementados para arquitetura de armazenamento, assim como, comparou-se os tempos de inserção de dados convencionais, geográficos e multimídia nos bancos de dados PostgreSQL, CouchDB e MongoDB, utilizando implementações em um único servidor, servidores replicados e servidores fragmentados. A ideia das provas de conceito é avaliar se a adoção dos bancos de dados NoSQL baseados em documento é

realmente uma alternativa viável para implementação da arquitetura de armazenamento de dados de um Sistema de Informação Geográfica Voluntária.

## 1.1 Objetivos

O objetivo geral deste trabalho é propor e desenvolver uma Arquitetura de Armazenamento de Dados para Sistemas de Informação Geográfica Voluntária utilizando Banco de Dados NoSQL baseado em Documento, que suporte os requisitos de escalabilidade e heterogeneidade dos dados.

No intuito de atingir o objetivo geral deste trabalho, foram definidos alguns objetivos específicos:

- Implementar um simulador para armazenamento de dados convencionais, geográficos e multimídia.
- Realizar análise de desempenho dos bancos de dados adotados para validação da arquitetura proposta.
- Analisar o comportamento do servidor de banco de dados mediante falhas nas máquinas das implementações distribuídas da arquitetura.
- Validar a arquitetura proposta através da implementação e testes de um protótipo de SIGV, em ambiente local e em *cluster*.

## 1.2 Estrutura do Trabalho

Este documento está estruturado nos capítulos a seguir:

- Capítulo 2, que apresenta a fundamentação teórica necessária para o desenvolvimento da pesquisa e alguns trabalhos relacionados, que compreendem os temas de Sistemas de Informação Geográfica, Sistemas de Informação Geográfica Voluntária, Sistemas de Banco de Dados Geográficos e Sistemas de Banco de Dados NoSQL.
- Capítulo 3, que apresenta a arquitetura proposta para o armazenamento de dados em Sistemas de Informação Geográfica Voluntária utilizando banco de dados NoSQL baseado em documento.
- Capítulo 4, que discute as provas de conceito da proposta através da implementação e validação da arquitetura de armazenamento de dados utilizando aplicativos móveis e banco de dados NoSQL, comparando-os a uma arquitetura de armazenamento tradicional que adota banco de dados relacional.

- Capítulo 5, que faz a conclusão da pesquisa através dos resultados obtidos nos testes realizados nas provas de conceito e aponta os trabalhos futuros.

# Capítulo 2

## Fundamentação Teórica

Este capítulo apresenta a fundamentação teórica necessária para possibilitar um melhor entendimento dos assuntos abordados neste trabalho. O presente capítulo está organizado da seguinte forma: a Seção 2.1 apresenta os principais conceitos, características e componentes dos SIG; a Seção 2.2 dedica-se a apresentar conceitos, desafios e aplicações dos SIGV; a Seção 2.3 aborda conceitos, características e exemplos de banco de dados que podem ser utilizados para armazenar dados espaciais; a Seção 2.4 apresenta conceitos, características, principais tipos de sistemas de banco de dados NoSQL e aplicações; e finalmente, a Seção 2.5 discute trabalhos que utilizam bancos de dados NoSQL para o armazenamento de dados convencionais e não convencionais, como dados espaciais. Para finalizar, a Seção 2.5 mostra trabalhos sobre SIGV e faz uma diferenciação entre os trabalhos relacionados e a arquitetura proposta.

### 2.1 Sistemas de Informação Geográfica

Na literatura existe algumas definições para Sistemas de Informação Geográfica (SIG), dentre as quais tem-se:

Os SIG podem ser definidos como um conjunto de procedimentos, manuais ou automatizados, utilizados no armazenamento e manipulação de informação georreferenciada [5].

Um SIG é um sistema de apoio à decisão que envolve a integração de dados espacialmente referenciados em um ambiente para resolução de problemas [16].

O termo Sistemas de Informação Geográfica é aplicado para sistemas que realizam o tratamento computacional de dados geográficos, mas que também possuem a capacidade de armazenar atributos descritivos (dados alfanuméricos) [12].

A partir destes conceitos, indica-se que as principais características de SIG são as seguintes [12]:

- Integrar, em uma base de dados, informações espaciais provenientes de meio ambiental, de dados censitários, de cadastros urbano e rural, e outras fontes de dados como imagens de satélite, e GPS.
- Oferecer mecanismos para combinar várias informações, através de algoritmos de manipulação e análise, bem como para consultar, recuperar e visualizar o conteúdo da base de dados geográficos.

Um SIG compreende três elementos básicos que operam em um contexto institucional: *hardware*, *software* e dados. Em muitos aspectos os dados são recursos essenciais, e é comum que o custo de aquisição dos dados ultrapasse o custo de *hardware* e *software*. O custo de aquisição e manutenção de dados espaciais equivale a aproximadamente 70% a 80% do custo total de alguns projetos de SIG [38, 40].

Segundo Casanova *et al* [12], os componentes presentes em um SIG são (Figura 2.1):

- Interface: proporciona interação do usuário com o sistema.
- Entrada e integração de dados: realiza a inserção de dados, incluindo mecanismos de conversão de dados.
- Consulta e análise espacial: utiliza algoritmos de consulta e análise espacial que incluem operações topológicas, álgebra de mapas, estatística espacial, modelagem numérica de terreno, e processamento de imagens.
- Visualização e plotagem: fornece suporte adequado para visualização e interpretação dos aspectos relevantes dos dados geográficos.
- Gerência dos dados espaciais: permite armazenamento e recuperação de dados espaciais e seus atributos.

A Figura 2.1 mostra a organização hierárquica e o relacionamento entre os componentes de um SIG. Como pode ser observado, a interface comunica-se com os componentes de entrada e integração de dados; consulta e análise espacial; e visualização e plotagem. Estes, por sua vez, comunicam-se diretamente com o componente de gerência de dados espaciais que, em seguida, comunica-se com o banco de dados geográfico responsável pelo armazenamento e recuperação de dados espaciais e outros atributos.

Existem basicamente duas formas de integração entre os SIG e os Sistema Gerenciador de Banco de Dados (SGBD), que são as arquiteturas dual e integrada [12]. Na arquitetura dual, os atributos não espaciais são armazenados em forma de tabelas em Sistema Gerenciador de Banco de Dados Relacional (SGBDR); e arquivos, em formatos proprietários, guardam as representações geográficas associadas aos registros das tabelas.

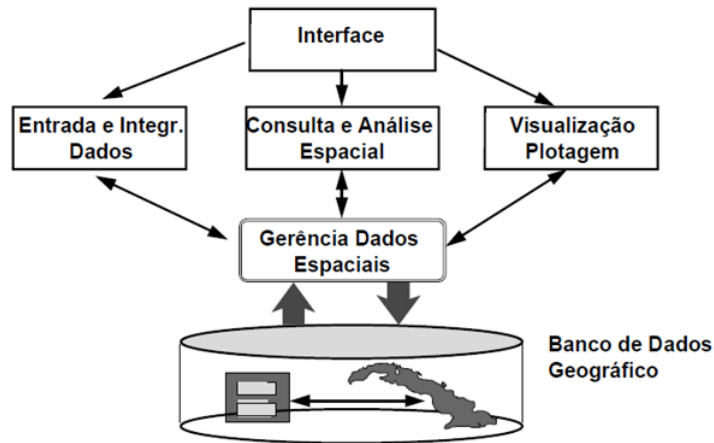


Figura 2.1: Arquitetura de Sistemas de Informação Geográfica [12].

Já na arquitetura integrada, tanto os dados convencionais quanto os dados espaciais são armazenados em um SGBD, tendo como principal vantagem a utilização dos recursos do SGBD para controlar e manipular dados espaciais, assim como realizar a gerência de transações, controle de integridade e concorrência.

De acordo com Casanova *et al* [12], há duas alternativas para a arquitetura integrada:

- Baseada em SGBD relacional: que utiliza campos longos do tipo binário - *Binary Large Object* (BLOB) - para armazenar os dados geográficos.
- Baseada em extensões espaciais sobre Sistema Gerenciador de Banco de Dados Objeto Relacional (SGBDOR): que estendem o modelo relacional fornecendo tipos de dados espaciais, como ponto, linha e polígono; e possibilita a execução de consultas SQL sobre esses tipos de dados não convencionais.

## 2.2 Sistemas de Informação Geográfica Voluntária

Os Sistemas de Informação Geográfica Voluntária (SIGV) são um tipo de SIG, que aproveitam da informação coletiva para aprimorar os dados disponíveis nos aplicativos. A interatividade da Internet permite que, através de um sistema dinâmico, organizações e usuários voluntários desempenhem papéis de consumidores e provedores de informação.

O termo Sistemas de Informação Geográfica Voluntária foi apresentado inicialmente em Goodchild [26], que notou uma mudança significativa na forma de aquisição e manutenção de dados geográficos a partir da colaboração de usuários voluntários.

Tradicionalmente, a aquisição de dados geográficos é realizada por especialistas bem treinados, que fazem o uso de tecnologias e métodos para coleta de dados sobre fenômenos

sociais e ambientais presentes na superfície terrestre como fotogrametria, sensoriamento remoto, redes de sensores, levantamentos topográficos, entre outros [13].

A abordagem de captura dos dados dos SIGV difere da abordagem dos dados espaciais tradicionais, que tendem a ser construídos por empresas e possuir uma estrutura bem definida. Já os dados dos SIGV podem ser heterogêneos e vindos de diversas fontes.

As principais características dos SIGV são [13]:

- Usuários registrados definem políticas e diretrizes.
- O acesso à rede de dados geográficos é bidirecional, ou seja, pode haver produção e recuperação de informação.
- Padrões específicos para cada caso são descritos pelo conteúdo dos dados.
- Os dados estão relacionados à um tema ou assunto específico.
- A massiva utilização do SIGV, pode ocasionar muitas operações de leitura e gravação, além de novos cadastros de usuários.

Desenvolvimentos recentes, como a plataforma web 2.0 e tecnologias móveis como *smartphones* equipados com receptores GPS (*Global Positioning System*); e redes de sensores, tornaram a tarefa de captura de dados geográficos não exclusiva aos especialistas treinados, e abriram novas oportunidades para o engajamento de cidadãos [17]. Por sua vez, acredita-se que o ser humano é capaz de capturar informações geográficas sobre fenômenos ambientais e/ou sociais, e em seguida, compartilhar essas informações a outros usuários através da internet [27].

Devido a sua natureza coletiva, os dados criados a partir de um SIGV tem sua credibilidade questionável [24], já que usuários não profissionais e pessoas mal intencionadas podem postar informação incorreta ou falsa, devendo os sistemas possuir mecanismos de verificação e validação das contribuições realizadas por usuários voluntários.

Há diversas discussões sobre a qualidade dos dados geográficos voluntários, principalmente no que diz respeito à sua precisão (acurácia) e validade (credibilidade) [24]. A avaliação da qualidade da informação geográfica segue um conjunto de medidas e critérios. A ISO (*International Organization for Standardization*), organização internacional para padronização, descreve padrões para qualidade dos dados (ISO 19157) e metadados especiais (ISO 19115) [42, 28].

Os parâmetros que definem a qualidade dos dados são completude, consistência lógica (integridade), precisão posicional, precisão temporal e precisão temática (adequação ao uso) [47]. Metadados sobre a qualidade dos dados são importantes para os SIGV, pois alguns cientistas desconsideram os dados voluntários por carecerem de tais informações [42, 3].



Mesmo diante dos riscos, a estratégia de utilizar mão de obra voluntária em SIGs é válida, pois a obtenção e manutenção de dados geográficos é uma tarefa dispendiosa e cara [40]. Além disso, é possível obter uma visão bem particular do sentimento dos cidadãos quanto a determinados assuntos políticos e sociais. Pode-se dizer então que o cidadão age como um “sensor humano participativo” no processo de produção de dados. Assim, a adoção de um SIGV pode ser uma alternativa para redução de custos e para atender às necessidades de indústrias, governos, comunidades e redes sociais [44].

O SIGV está relacionado com diferentes interesses, de acordo com o público alvo. Sob o olhar da indústria, o interesse em SIGV está relacionado ao desenvolvimento de ferramentas apropriadas para o gerenciamento de dados espaciais, com a possibilidade de propor maneiras de verificar e validar os dados geográficos voluntários. O interesse do governo pode ser a adoção de políticas públicas que busquem sanar problemas locais, como mapeamento de doenças, crimes, níveis de ruído; melhoria do saneamento em áreas necessitadas; provimento de ajuda em casos de calamidades; entre outras utilizações em benefício do planejamento urbano. Sob o olhar acadêmico, existem campos de pesquisa que se preocupam com a qualidade dos dados, visualização das informações geográficas, melhores maneiras de lidar com grandes volumes de dados heterogêneos, assim como a informação pode ser armazenada, gerenciada, pesquisada e compartilhada de forma eficiente [47, 23].

No contexto de Sistemas de Informação Geográfica Voluntária alguns projetos se destacam, dentre eles: Wikicrimes<sup>1</sup>, Wikimapia<sup>2</sup> e OpenStreetMap<sup>3</sup>. A seguir uma descrição de cada um deles é apresentada.

### 2.2.1 Wikicrimes.org

Wikicrimes<sup>4</sup> é um Sistema de Informação Geográfica Voluntária que permite acompanhar e registrar ocorrências criminais em um mapa interativo. O SIGV pode ser utilizado através de um computador ou dispositivo móvel das plataformas iOS ou Android. O Wikicrimes segue uma filosofia colaborativa semelhante a enciclopédia Wikipedia, partindo do princípio que a colaboração individual pode gerar conhecimento a um grupo de pessoas.

O objetivo deste SIGV é alertar a população sobre locais inseguros e fazer com que as pessoas fiquem mais atentas ao frequentar esses locais. A documentação do sistema deixa claro que em nenhum momento o projeto deseja substituir o registro formal dos crimes nas unidades policiais, mas salienta que todos podem tirar proveito das informações mapeadas pelos usuários voluntários.

---

<sup>1</sup><http://wikicrimes.org>

<sup>2</sup><http://wikimapia.org/>

<sup>3</sup><http://openstreetmap.org/>

<sup>4</sup><http://wikicrimes.org>

A Figura 2.2 apresenta a interface do SIGV Wikicrimes, onde é possível visualizar a densidade de crimes praticados nas regiões marcadas no mapa. Essa densidade é classificada como baixa, média e alta incidência de crimes e pode ser facilmente visualizada por círculos de diferentes cores.

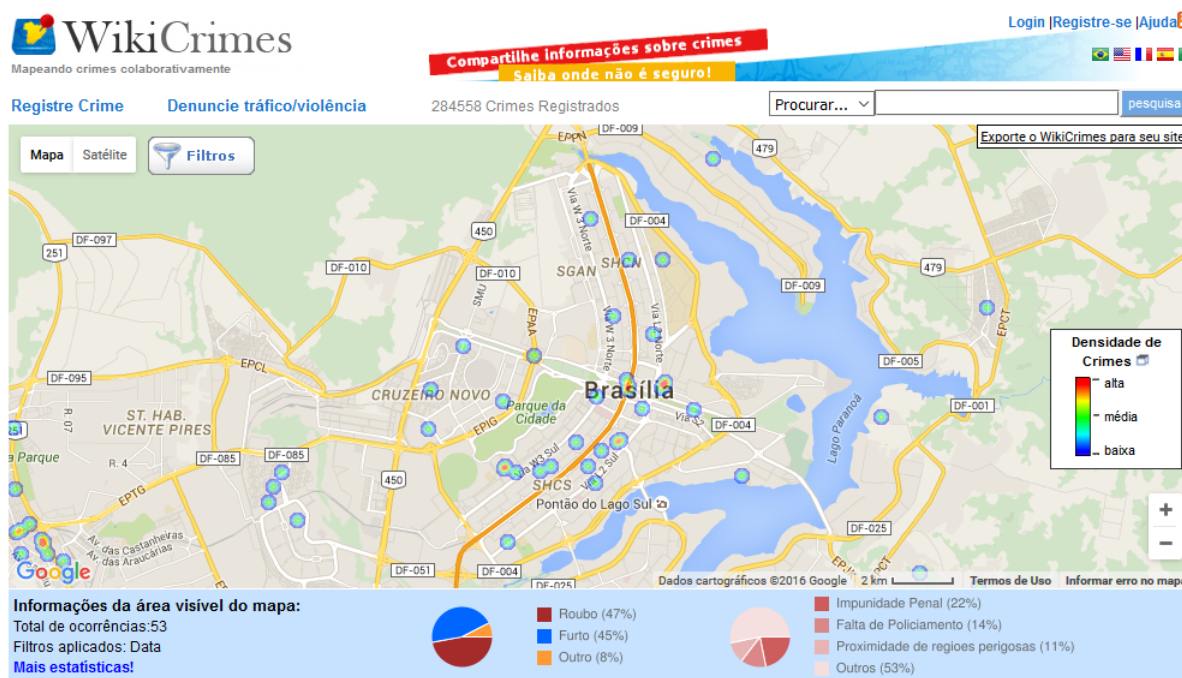


Figura 2.2: Interface do software Wikicrimes.org.

## 2.2.2 Wikimapia.org

Wikimapia<sup>5</sup> é um projeto de mapeamento colaborativo destinado a marcar objetos geográficos do mundo para fornecer uma descrição útil a eles. O SIGV também pode ser utilizado através de um computador ou dispositivo móvel das plataformas iOS ou Android, e, assim como o Wikicrimes, combina um mapa interativo com o sistema wiki.

O objetivo deste SIGV é descrever o mundo através da coleta de informações úteis sobre objetos geográficos, organizá-las e disponibilizá-las para reutilização sob licença de compartilhamento *Creative Commons*.

O SIGV foi desenvolvido com o intuito de ser simples, podendo ser utilizado por qualquer pessoa, mesmo as que não tem experiência com mapas. Segundo a documentação do projeto, uma das principais características do Wikimapia é que ele é frequentemente atualizado através das contribuições de seus usuários voluntários.

<sup>5</sup><http://wikimapia.org/>

A Figura 2.3 apresenta o *software* Wikimapia, onde é possível buscar e compartilhar informações úteis sobre pontos de interesse através de um mapa interativo. O usuário da aplicação pode filtrar a exibição desses pontos por categoria para facilitar o acesso aos objetos geográficos presentes no mapa. O Wikimapia ainda conta com uma ferramenta de medição de distância entre pontos, que pode ser útil em determinadas situações.



Figura 2.3: Interface do software Wikimapia.org.

### 2.2.3 OpenStreetMap.org

O OpenStreetMap<sup>6</sup> é um SIGV de mapeamento colaborativo que tem o intuito de criar um mapa livre e editável do mundo sob uma licença aberta. Os mapas do projeto são criados a partir do trabalho de uma comunidade de usuários voluntários que utilizam dados de dispositivos GPS, fotografias aéreas e outras fontes de informação para contribuir e manter atualizados dados sobre áreas urbanas, rodovias, ferrovias, parques, lagos, rios, entre outros itens de interesse geográfico.

A Figura 2.4 exibe a interface do projeto OpenStreetMap, que fornece dados de mapas para *sites* e aplicações de celular e outros dispositivos.

## 2.3 Bancos de Dados Geográficos

Os Bancos de Dados Geográficos, também chamados de Bancos de Dados Espaciais, são semelhantes aos bancos de dados relacionais, porém suportam formas geométricas georreferenciadas (dados geoespaciais). Os dados geoespaciais podem ser categorizados

<sup>6</sup><http://openstreetmap.org/>

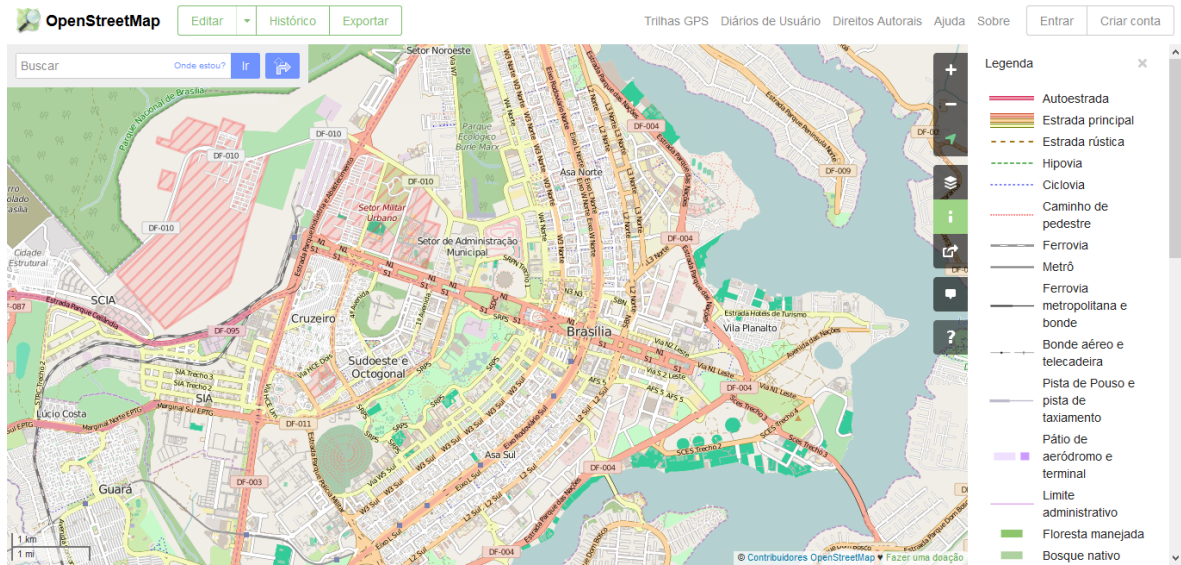


Figura 2.4: Interface do projeto OpenStreetMap.org.

de duas formas: dados *raster* e dados vetoriais. Os dados *raster* podem ser geomagens obtidas a partir de veículos aéreos não tripulados (*drones*), câmeras de segurança e/ou satélites. Os dados vetoriais consistem de formas geométricas como pontos, linhas e polígonos. Trabalhando com objetos geométricos, esses sistemas possibilitam consultas e análises espaciais [10, 51, 34].

A forma de armazenamento computacional de dados geográficos e a maneira de relacionar essas estruturas geométricas e alfanuméricas com dados do mundo real são bastante discutidas. A representação computacional do espaço geográfico é considerada o problema fundamental da geo-informação. De acordo com Câmara *et al* [10], para abordar esse problema utiliza-se o paradigma dos quatro universos [12].

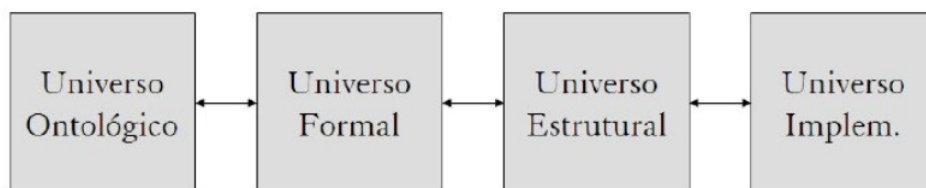


Figura 2.5: Paradigma dos quatro universos [12].

A Figura 2.5 apresenta o paradigma dos quatro universos, que são quatro passos para realizar a representação computacional de dados geográficos. O primeiro passo – universo ontológico – inclui os conceitos dos objetos que serão representados no computador, como tipo de solo, vegetação e caracterização das formas do terreno. O universo ontológico trabalha a geo-ontologia que é um conjunto de conceitos e relações semânticas e espaciais dos objetos do mundo real. Para representação de ontologias geográficas, o

consórcio OGC (*Open Geospatial Consortium*) propôs o formato GML (*Geography Markup Language*). No segundo passo – universo formal – têm-se as representações lógicas ou matemáticas que criam abstrações formais que dão base ao universo ontológico. Essas abstrações incluem modelos de dados e álgebras computacionais. No terceiro passo – universo estrutural – as entidades dos modelos formais são mapeadas para estruturas de dados geométricas e alfanuméricas, e algoritmos que realizam operações. O quarto e último passo – universo de implementação – finaliza o processo de representação computacional realizando a implementação do sistema e selecionando arquiteturas, linguagens e paradigmas de programação [12].

A representação computacional das entidades geográficas necessita de modelos formais como geo-campos, geo-objetos e de redes. Os dois primeiros referem-se a modelos do espaço absoluto, e o último a um modelo do espaço relativo. O espaço absoluto trata-se da representação da localização de objetos através de coordenadas geográficas; e o espaço relativo refere-se principalmente as relações de adjacências entre os objetos [12].

O modelo de geo-campos visualiza o espaço geográfico como uma superfície contínua, na qual variam os fenômenos a serem observados. Por exemplo, em um mapa de vegetação, cada ponto representa um tipo específico de cobertura vegetal. O modelo de geo-objetos representa o espaço geográfico como uma coleção de objetos distintos e identificáveis, onde cada objeto possui uma fronteira fechada. Por exemplo, em um cadastro urbano, pode-se representar cada lote como sendo um objeto distinto e indivisível, possuidor de atributos que o diferencia dos demais. O modelo de redes representa o espaço geográfico como um conjunto de pontos (nós) conectados a linhas (arestas), onde tanto os nós quanto às arestas podem possuir atributos descritivos. O modelo de redes tem utilidade em aplicações de gerenciamento de infraestruturas de serviços, tais como água e esgoto, eletricidade e telefonia [12].

O modelo formal básico para dados geográficos baseia-se nos conceitos de geo-campo, coleção de geo-objetos e rede, e é chamado de modelo orientado a objetos. A Figura 2.6 representa a organização lógica do modelo, considerando a existência de uma classe genérica chamada de plano de informação (ou *layer*) que captura características comuns dos três conceitos básicos, que são a existência de uma localização no espaço e de um identificador único. Cada geo-campo pode ser especializado em geo-campo temático, referente a medidas nominais ou ordinais; ou em geo-campo numérico, associado às medidas por intervalo.

As estruturas de dados associadas aos banco de dados geográficos são classificadas em estruturas vetoriais e estruturas matriciais. As estruturas vetoriais (Figura 2.7(a)) são utilizadas para representar as coordenadas cartesianas de cada objeto geográfico, como ponto, linha ou polígono. Um ponto é um par  $(x, y)$  de coordenadas espaciais; uma linha

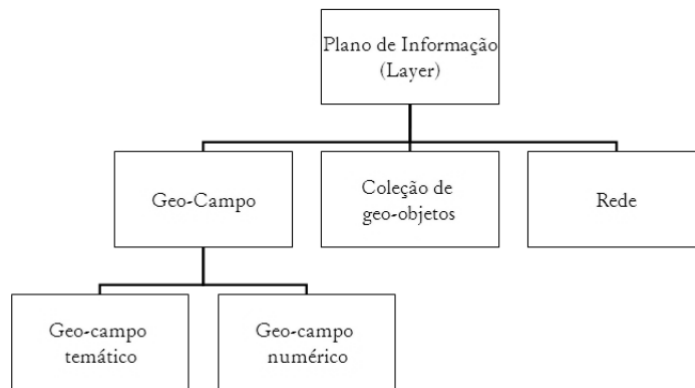


Figura 2.6: Modelo orientado a objetos básicos para dados geográficos [12].

é um conjunto de pontos conectados; e um polígono é uma região delimitada por pontos conectados que formam um circuito fechado. As estruturas matriciais (Figura 2.7(b)) representam o espaço como uma superfície plana, organizada em uma grade regular, onde cada célula representa uma porção do espaço. Na representação matricial o espaço é representado por uma matriz  $P(m, n)$ , onde  $m$  é o número de colunas e  $n$  é o número de linhas, sendo que cada célula pode ser acessada individualmente por suas coordenadas [12]. Nesta dissertação, somente dados vetoriais são tratados.

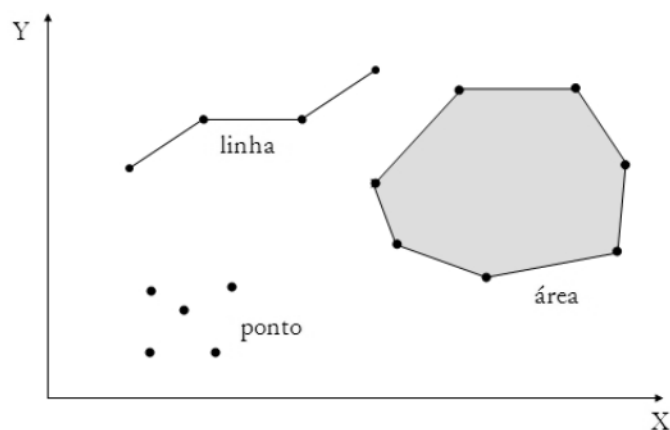
Um modelo de dados é um conjunto de conceitos que podem ser usados para descrever a estrutura de um banco de dados. Estrutura de um banco de dados inclui tipos de dados, relacionamentos e as restrições que se aplicam aos dados. A modelagem de dados geográficos é uma atividade complexa, pois envolve a discretização do espaço como parte do processo de abstração, visando obter representações adequadas aos fenômenos geográficos [22, 10]. Citam-se algumas técnicas de modelagem para Banco de Dados Espaciais como OMTG [7], MODUL-R [9], GMOD [48], GISER [53], GeoOOA [33] e OMT EXT [2].

Os Sistemas Gerenciadores de Bancos de Dados tradicionais, tais como Oracle<sup>7</sup>, PostgreSQL<sup>8</sup> e MySQL<sup>9</sup>, não suportam nativamente dados geográficos, ou seja, formas geométricas. No entanto, é possível instalar e executar extensões adicionais que possibilitem que esses dados sejam armazenados e manipulados através de consultas e análises espaciais. O PostGIS é um exemplo dessas extensões adicionais, que tem o objetivo de fornecer funções de banco de dados espaciais ao PostgreSQL. Tem-se outros exemplos de extensões espaciais como MySQL *Spatial* e Oracle *Spatial* [56].

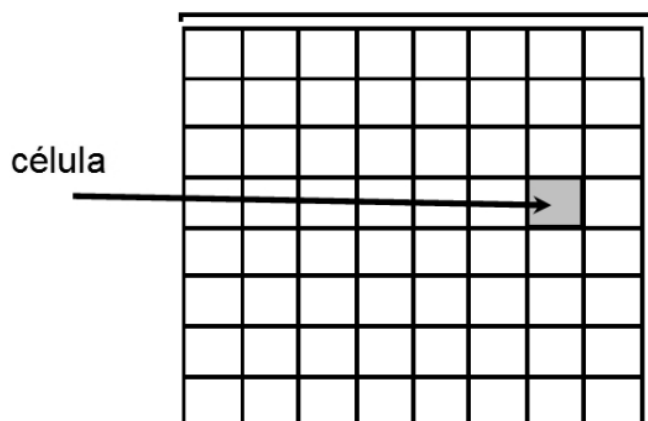
<sup>7</sup><http://www.oracle.com/br/database/overview/index.html>

<sup>8</sup><https://www.postgresql.org/>

<sup>9</sup><http://www.mysql.com/>



(a) Vetores em duas dimensões.



(b) Estrutura matricial.

Figura 2.7: Representação vetorial de objetos geográficos e estrutura matricial de representação do espaço [12].

## 2.4 Sistemas de Banco de Dados NoSQL

Os sistemas de banco de dados NoSQL surgiram da necessidade de armazenamento de grandes quantidades de dados, principalmente, em aplicações *web*. As aplicações *web* podem apresentar utilização de um grande número de usuários simultâneos, o que pode comprometer o desempenho do SGBD. As soluções NoSQL são projetadas para atender a requisitos de disponibilidade e escalabilidade sob demanda, visto as necessidades de aplicações de larga escala. Pode-se dizer então que os sistemas de banco de dados NoSQL são um produto da era *web 2.0*, já que são úteis para tratar uma enorme quantidade de dados e usuários, e os casos em que as características dos dados não requer um modelo relacional [45, 58].

A publicação dos artigos do Google Bigtable [14], em 2006, e do Amazon Dynamo [20], em 2007, inspirou a criação de vários projetos de Banco de Dados que faziam ex-

permutações com armazenamentos alternativos de dados, como o Riak<sup>10</sup>, MongoDB<sup>11</sup>, HBase<sup>12</sup>, Cassandra<sup>13</sup>, e Neo4j<sup>14</sup>, foram todos criados entre os anos de 2007 e 2009.

Acredita-se que o termo NoSQL, da maneira que é conhecido atualmente, é resultado de uma reunião realizada no dia 11 de junho de 2009, em São Francisco, nos Estados Unidos, quando houve algumas palestras sobre bancos de dados não relacionais, distribuídos e de código aberto. O NoSQL refere-se a um sistema de banco de dados projetado para ambientes computacionais distribuídos, que não requer um esquema fixo de tabelas, que possui escalabilidade horizontal e tem a capacidade de superar problemas emergentes em armazenamento de dados, tais como volume, velocidade e variedade [52, 25, 58, 41].

A escalabilidade horizontal é uma tentativa de melhorar o desempenho do sistema através do aumento do número de unidades de processamento e armazenamento. Neste caso, cria-se um *cluster* de computadores que, interligados e trabalhando em conjunto, podem exceder a capacidade de processamento de um único nó de armazenamento [55].

Não há uma definição genericamente aceita para os NoSQL, mas podem-se citar algumas características que são comuns a esses Sistemas Gerenciadores de Banco de Dados [52, 31, 46]:

- Modelo não Relacional: os NoSQL não utilizam o modelo relacional.
- Modelo de dados sem esquema definido: permite maior flexibilidade no armazenamento dos dados uma vez que alguns NoSQL não requerem esquemas.
- Projetado para processamento distribuído (em *clusters*) e escalabilidade horizontal: organizações obtém cada vez mais dados e para processá-los mais rapidamente é interessante que se utilize um *cluster* de computadores.
- Suporte ao armazenamento de dados em massa e concorrentes operações de leitura e escrita: a mudança na arquitetura do sistema permite que a aplicação tenha um maior poder de processamento e armazenamento de dados.
- Não utiliza a SQL como linguagem de consulta: alguns dos bancos de dados possuem linguagens de consulta semelhantes ao SQL para que sejam mais facilmente aprendidas.
- Geralmente são de código aberto: embora o termo NoSQL também seja aplicado a sistemas de código fechado.

---

<sup>10</sup><http://basho.com/products/>

<sup>11</sup><https://www.mongodb.com/>

<sup>12</sup><https://hbase.apache.org/>

<sup>13</sup><http://cassandra.apache.org/>

<sup>14</sup><https://neo4j.com/>



- Baixa aderência ao modelo de consistência ACID (Atomicidade, Consistência, Isolamento e Durabilidade): havendo processamento distribuído é possível obter respostas rápidas às consultas, porém é necessário relaxar a exigência da consistência ou das propriedades ACID, comuns no processamento de transações em bancos de dados relacionais.

Os bancos de dados NoSQL possuem princípios derivados do teorema CAP [29, 8]. De acordo com este teorema, as seguintes garantias podem ser definidas:

- Consistência (*Consistency*): todos os nós participantes do *cluster* têm os mesmos dados ao mesmo tempo.
- Disponibilidade (*Availability*): todas as requisições têm resposta.
- Tolerância à Partição (*Partition tolerance*): o sistema continua funcionando mesmo havendo particionamentos na rede.

Em um sistema de banco de dados distribuído, é possível garantir até dois elementos do teorema CAP. A consistência é a característica que descreve o estado do sistema após determinada operação. A disponibilidade refere-se a capacidade do sistema se recuperar de falhas no *hardware/software* para se manter em funcionamento. A tolerância a partição refere-se a capacidade de realizar operações de leitura e escrita mesmo após serem realizados diversos particionamentos na rede. [29, 8].

Os sistemas NoSQL seguem os princípios BASE, que são caracterizados pela alta disponibilidade dos dados e que, por consequência, sacrificam momentaneamente a consistência [1]:

- Basicamente disponível (*Basically Available*): todos os dados estão distribuídos, mesmo havendo uma falha o sistema continua a funcionar.
- Estado simples (*Soft state*): não precisa ser consistente o tempo todo.
- Eventualmente consistente (*Eventually Consistent*): o sistema garante que mesmo que os dados não estejam consistentes, eventualmente eles serão.

### 2.4.1 Escalabilidade, Replicação e *Sharding* de Bancos de Dados NoSQL

Os sistemas de banco de dados NoSQL abordam conceitos de escalabilidade, replicação e *sharding*. Cada uma dessas características são apresentadas nesta Seção.

## Escalabilidade

Segundo Elmasri *et al* [22], a escalabilidade determina a extensão que um sistema pode aumentar sua capacidade e manter em funcionamento sem interrupção. Existem dois tipos de escalabilidade [57]:

- Escalabilidade horizontal: que significa aumentar o número de nós (máquinas) no sistema distribuído, tendo como benefício a facilidade na distribuição dos dados e balanceamento de carga por várias máquinas. Neste caso, é possível utilizar *hardware* comum e de menor custo.
- Escalabilidade vertical: que significa aumentar a capacidade do *hardware* do servidor, sendo uma escalabilidade limitada e com custos elevados.

A escalabilidade horizontal é uma das principais características dos sistemas de bancos de dados NoSQL. Isto permite que um grande volume de operações de leitura/escrita sejam executadas muito mais eficientemente [61].

## Replicação

A replicação é uma maneira de criar cópias idênticas das informações do banco de dados em vários servidores e é recomendada para todas as instalações em produção, ou seja, recomenda-se para todas as aplicações que passaram do período de testes e são utilizadas por usuários reais. Com várias cópias dos dados em diferentes servidores de banco de dados, é possível fornecer um nível de tolerância a falhas contra a perda de um servidor de banco de dados único. Assim, a replicação mantém o aplicativo em execução, mesmo que alguma falha aconteça com algum dos servidores disponíveis [15].

Em alguns casos, a replicação pode fornecer maior capacidade de leitura, pois os clientes do banco de dados podem enviar operações de leitura para diferentes servidores. Manter cópias de dados em diferentes *datacenters* pode melhorar o desempenho de resposta e disponibilidade de aplicações distribuídas [39].

Existem três tipos de arquitetura de replicação [52, 37]:

- *Master-slave* (mestre-escravo): nesta arquitetura ocorre a replicação dos dados em múltiplos nós, ou seja, a distribuição de cópias dos dados entre os nós participantes do *cluster* de computadores. Um nó é designado mestre, sendo a fonte oficial dos dados, e geralmente, tem a responsabilidade de processar quaisquer atualizações nesses dados. Os nós escravos são considerados secundários, e tem função de lidar com as leituras de dados do sistema. O processo de replicação sincroniza os dados do nó mestre com os dados dos nós escravos. A replicação mestre-escravo é mais útil

para a escalabilidade quando há um conjunto de dados com muitas leituras. Este tipo de arquitetura tem uma concepção simples, com boa consistência, mas pode dar origem a um único ponto de falha (nó mestre). No entanto, mesmo se o nó mestre falhar ainda é possível que as operações de leitura sejam realizadas. Além disso, é possível eleger um novo nó mestre a partir de um nó escravo para se recuperar da falha [52, 37]. Exemplos: MongoDB<sup>15</sup> (master-slave) e Neo4j<sup>16</sup> (master-slave estilo MySQL).

- *Master-master* (mestre-mestre): também conhecida como arquitetura multi-mestre, onde todos ou alguns dos nós do *cluster* possuem a capacidade de realizar operações de leitura e gravação. Nesta arquitetura a replicação dos dados ocorre de forma síncrona, buscando não haver discrepância nos dados entre quaisquer dos nós disponíveis. É importante destacar que é possível haver problemas de consistência, caso haja falhas de comunicação entre os nós. Esta arquitetura tenta ser uma alternativa mais flexível em relação a arquitetura mestre-escravo. Exemplo: CouchDB<sup>17</sup>.
- *Peer-to-Peer* (ponto a ponto): nesta arquitetura, as réplicas têm peso igual, assim todos os nós do *cluster* podem receber gravações, e a falha de algum dos nós não impede o acesso ao armazenamento de dados. Esta arquitetura é altamente escalável, sendo possível adicionar nós para melhorar o desempenho do sistema. No entanto, há preocupações principalmente em relação à consistência dos dados, já que é possível haver conflitos de gravação. É recomendável utilizar este tipo de arquitetura quando a aplicação tiver foco na gravação de dados, e seja necessário priorizar o requisito de disponibilidade à consistência [52, 37]. Exemplo: Cassandra.

## Sharding

*Sharding* ou estilhaçamento (tradução livre) é uma técnica que realiza a fragmentação de diferentes partes dos dados em diversos servidores do *cluster* do banco de dados. Esta técnica consiste em dividir os dados em fragmentos independentes pelos nós existentes, o que faz que quando um cliente necessite dos dados seja redirecionado para o nó que os contém [52, 57].

A proposta do *sharding* é equilibrar a carga dos dados entre os servidores do banco de dados, fazendo com que os dados se tornem mais disponíveis e que os usuários obtenham respostas mais rápidas [50]. Citam-se algumas razões para utilizar essa técnica no banco de dados [57]:

---

<sup>15</sup><https://www.mongodb.com/>

<sup>16</sup><https://neo4j.com/>

<sup>17</sup><http://couchdb.apache.org/>

- Os dados podem ser colocados geograficamente mais próximo do usuário.
- Consultas podem ter um melhor desempenho já que há redução do tamanho do conjunto de dados.
- Organiza os fragmentos muito utilizados em servidores diferentes para equilibrar a carga do sistema.

Para obter um melhor desempenho, deve-se garantir que os dados que serão acessados na mesma consulta estejam armazenados no mesmo nó. As técnicas de modelagem de agregados podem contribuir para que isso seja possível [52].

## 2.4.2 Tipos de Banco de Dados NoSQL

Atualmente, existem mais de 225 bancos de dados NoSQL catalogados<sup>18</sup>, sendo eles classificados de acordo com o modelo de dados utilizado. Tipicamente os bancos de dados NoSQL são classificados em quatro categorias [31, 22]: chave-valor (*Key-Value Store*), documento (*Document Store*), coluna (*Column Store*) e grafo (*Graph Store*).

As três primeiras categorias possuem uma característica comum em seus modelos de dados, que pode ser chamada de orientação agregada. A orientação agregada considera que os dados são trabalhados na forma de unidades, e que tenham uma estrutura mais complexa que um conjunto de tuplas. Um agregado é um conjunto de objetos associados que se deseja tratar como uma unidade de manipulação de dados e gerenciamento de consistência. Lidar com agregados facilita a execução do banco de dados em um *cluster*, uma vez que o agregado constitui uma unidade natural para replicação e fragmentação [52].

Os agregados também tem uma consequência importante para transações. Muitas vezes se diz que bancos de dados NoSQL não suportam transações ACID, no entanto, é possível suportar manipulação atômica em um único agregado por vez. Os bancos de dados baseados em grafo e outros bancos de dados não agregados, geralmente, suportam transações ACID de modo semelhante aos bancos de dados relacionais [52].

Deve-se analisar a adoção de um determinado tipo de banco de dados NoSQL, de acordo com as características da aplicação e dos dados que serão armazenados.

### Banco de Dados baseado em Chave-Valor (*Key-Value Store*)

Nestas bases de dados, as informações são armazenadas como um conjunto de chave-valor. Todas as chaves tem nomes únicos e o acesso aos dados é feito relacionando às chaves aos valores. Um vetor *hash* contém todas as chaves a fim de prover informação

---

<sup>18</sup><http://nosql-database.org/>

quando necessário. Neste caso, somente as chaves podem ser pesquisadas pelos usuários do banco de dados [1].

O banco de dados baseado em chave-valor pode ser comparado a um dicionário. Um dicionário tem uma lista de palavras e cada palavra tem uma ou mais definições. As palavras representam as chaves, e as definições os valores [41].

Considera-se que os bancos de dados chave-valor são os depósitos de dados NoSQL mais simples de utilizar a partir de uma perspectiva de uma API (*Application Programming Interface*). O cliente pode obter um valor de uma determinada chave, inserir um valor para uma determinada chave ou apagar uma chave do depósito de dados. O valor é um campo do tipo BLOB que será armazenado sem a preocupação de saber o que há dentro dele; é responsabilidade do aplicativo entender o que foi armazenado [52]. Campos do tipo BLOB podem armazenar dados do tipo imagem, páginas web, documentos, vídeos, entre outros formatos [41].

A simplicidade do armazenamento chave-valor torna-o ideal para recuperação de valores em aplicativos que possuem perfis de usuário, gerenciamento de sessões ou recuperação de nomes de produtos, como carrinho de compras.

Alguns exemplos de bancos de dados baseados em chave-valor são DynamoDB<sup>19</sup>, Azure Table Storage<sup>20</sup>, Riak<sup>21</sup>, Redis<sup>22</sup> e Voldemort<sup>23</sup> [20].

## **Banco de Dados baseado em Coluna (*Column Store*)**

Os bancos de dados baseados em coluna têm organização similar ao modelo de banco de dados relacional. A estrutura de dados e organização consiste de coluna (representa a unidade do dado identificado por chave e valor), super coluna (agrupa as informações da coluna), e família de colunas (conjunto de dados estruturados - semelhante a tabela dos bancos de dados relacionais - constituída por uma variedade de super colunas). A estrutura do banco de dados é definida por super colunas e família de colunas. Novas colunas podem ser adicionadas quando necessário [1].

A melhor maneira de pensar no modelo de famílias de colunas é como uma estrutura agregada em dois níveis. No primeiro nível, tem-se uma chave que funciona como um identificador de linha, buscando o item (agregado) de interesse. O agregado de linha é formado por um conjunto com valores mais detalhados. Esses valores de segundo nível são chamados de colunas. Além de acessarem a linha como um todo, as operações também permitem a seleção de uma coluna em particular [52].

---

<sup>19</sup><https://aws.amazon.com/pt/dynamodb/>

<sup>20</sup><https://azure.microsoft.com/pt-br/>

<sup>21</sup><http://basho.com/>

<sup>22</sup><http://redis.io/>

<sup>23</sup><http://www.project-voldemort.com/voldemort/>

Recomenda-se utilizar o banco de dados baseado em coluna para o armazenamento de dados distribuídos, especialmente dados sob controle de versão; aplicações de grande porte, de processamento de dados orientado a lotes; e para análise preditiva e exploratória de dados estatísticos.

Alguns exemplos de bancos de dados baseados em coluna são Hbase<sup>24</sup>, Cassandra<sup>25</sup>, Hypertable<sup>26</sup> e Amazon SimpleDB<sup>27</sup> [45].

### **Banco de Dados baseado em Grafo (*Graph Store*)**

Os bancos de dados baseados em grafo fundamentam-se na Teoria dos Grafos e fornece três construtores básicos: nós (ou vértices), ligações (ou arestas) e propriedades. Os nós são usados para modelar objetos que existem independentemente das ligações. Objetos (nós) de um mesmo grafo podem ter atributos distintos. As arestas permitem estabelecer relacionamento entre os nós, e as propriedades podem descrever os atributos de nós e arestas.

Bancos de dados baseados em grafo possuem registros pequenos com interconexões complexas. Embora bancos de dados relacionais possam implementar relacionamentos utilizando chaves estrangeiras, as junções necessárias para navegar por eles podem ser bastante custosas, o que significa que o desempenho é ruim para modelos de dados altamente conectados. Por outro lado, os bancos de dados baseados em grafo possuem foco em relacionamentos, e por isso, tendem a funcionar em um único servidor [52].

O uso do banco de dados baseado em grafo é indicado quando se necessita representar relacionamentos, como em redes sociais, detecção de fraudes e aplicações que exigem alto desempenho em consultas com muitas junções. Alguns exemplos de bancos de dados baseados em grafo são Neo4J<sup>28</sup>, Infinite Graph<sup>29</sup>, Sparksee<sup>30</sup>, TITAN<sup>31</sup> e InfoGrid<sup>32</sup> [1, 41].

### **2.4.3 Banco de Dados baseado em Documento (*Document Store*)**

Os bancos de dados baseados em documento armazenam dados com a estrutura de documentos com padrões reconhecidos, como XML (*eXtensible Markup Language*) ou JSON

---

<sup>24</sup><https://hbase.apache.org/>

<sup>25</sup><http://cassandra.apache.org/>

<sup>26</sup><http://www.hypertable.org/>

<sup>27</sup><https://aws.amazon.com/pt/simplydb/>

<sup>28</sup><https://neo4j.com/>

<sup>29</sup><http://www.objectivity.com/products/infinitegraph/>

<sup>30</sup><http://sparsity-technologies.com/>

<sup>31</sup><http://titan.thinkaurelius.com/>

<sup>32</sup><http://infogrid.org/trac/>

(*JavaScript Object Notation*). Neste tipo de armazenamento, todas as chaves e valores podem ser pesquisadas pelos usuários do banco de dados, fornecendo uma consulta consistente sobre os dados armazenados [25, 1].

O banco de dados baseado em documento é bastante flexível e não possui um esquema definido, podendo carregar qualquer tipo de documento, sem a necessidade de conhecer previamente a estrutura do documento a ser gerenciado pelo SGBD [25].

Bancos de dados baseados em documento podem armazenar dados estruturados, tais como arquivos CSV (*Comma-separated values*); semi estruturados, como arquivos XML e HTML (*HyperText Markup Language*); e não estruturados, como arquivos de imagem, áudio, vídeo, documentos PDF (*Portable Document Format*) e de texto. Alguns SGBDs deste tipo possuem a funcionalidade de extrair metadados de arquivos binários, o que pode ser interessante para indexação e para proporcionar uma gestão mais eficiente dos arquivos (documentos) armazenados [25].

Um documento é a unidade de dados básica dos bancos de dados baseados em documento, e pode ser considerado equivalente ao registro (linha) em um banco de dados relacional. Alguns bancos de dados baseados em documento tem sua estrutura organizada por coleção para oferecer gerenciamento unificado de um número de documentos, e várias coleções podem pertencer a mesma base de dados. As coleções são semelhantes às tabelas do modelo relacional, porém são constituídas por documentos e são livres de esquema para que documentos de diferentes estruturas possam ser armazenados em uma mesma coleção. Uma base de dados pode incluir inúmeras coleções e cada servidor de banco de dados pode ter múltiplas bases de dados com acesso independente [15, 62].

O Código 2.1 apresenta a estrutura de um documento simples com informações de um cliente, formatado na especificação JSON. Como pode-se observar o documento possui os seguintes atributos: id, nome, endereço e telefone. Geralmente, o conteúdo do documento é armazenado em modo chave-valor. A primeira chave é gerada automaticamente pelo sistema de banco de dados, usualmente com o nome "\_id", e funciona como identificador do documento no banco de dados. Os valores numéricos não são acompanhados por aspas, já os valores alfanuméricos necessitam das aspas em sua representação [22, 62].

---

```
1 {  
2   "_id": 1,  
3   "nome": "Joao Silva",  
4   "endereco": "Avenida das Acacias, 345",  
5   "telefone": "(61) 987229955"  
6 }
```

---

Código 2.1: Estrutura do documento JSON: Cliente (simples).

Um documento pode possuir uma estrutura mais complexa que um registro de uma tabela em um banco de dados relacional. É possível, por exemplo, representar detalhadamente um atributo composto utilizando um dicionário, que é uma estrutura de um documento utilizado como valor de uma chave. Há também a possibilidade de representar uma lista de dicionários. O Código 2.2 apresenta a estrutura do documento JSON de um cliente semelhante ao Código 2.1, porém, neste exemplo, têm-se a especificação do atributo composto “endereço” (linhas 4 a 7), e da lista de dicionários no campo “telefone” (linhas 8 a 16) [22].

---

```
1 {
2   "_id": 1,
3   "nome": "Joao Silva",
4   "endereco": {
5     "logradouro": "Avenida das Acacias",
6     "numero": 345
7   },
8   "telefone": [{
9     "ddd": "61",
10    "telefone": "987229955",
11    "tipo": "celular"
12  }, {
13    "ddd": "61",
14    "telefone": "74990033",
15    "tipo": "fixo"
16  }]
17 }
```

---

Código 2.2: Estrutura do documento JSON: Cliente (dicionários).

Os documentos também podem ser embutidos, ou seja, os documentos podem ser incorporados em um único documento a partir de um campo ou lista. Esse modelo desnormalizado permite que os dados sejam armazenados em uma única operação no banco de dados [60].

Em [60], apresenta-se um padrão para modelagem de dados em bancos de dados NoSQL baseados em documento. Essa técnica de modelagem é utilizada para representar graficamente (Figura 2.8) a interação dos documentos embutidos (cliente, pedido e produto) no Código 2.3 a seguir.

O Código 2.3 exemplifica essa organização e apresenta a estrutura de um documento JSON de um cliente que inclui uma lista de documentos embutidos de pedidos. Neste exemplo, cada pedido pode possuir uma lista de itens do pedido, que utiliza informações de outro documento (produto).

---

```
1 {
```



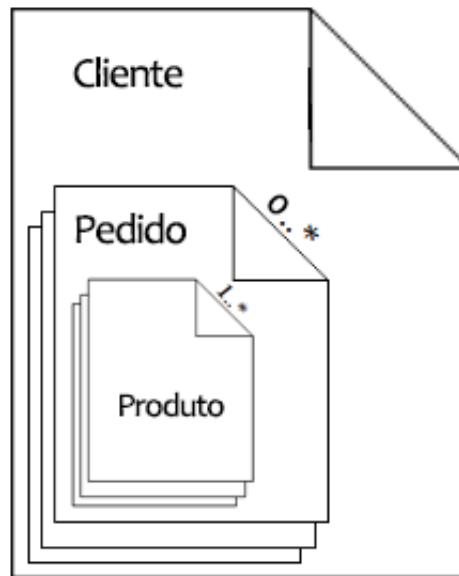


Figura 2.8: Representação do modelo não relacional do Código 2.3 utilizando notação de Vera *et al* [60].

```

2     "_id": 1,
3     "nome": "Joao Silva",
4     "endereco": "Avenida das Acacias, 345",
5     "pedidos": [{
6         "id": 421,
7         "clienteId": 1,
8         "itensPedidos": [{
9             "produtoId": 33,
10            "produtoDescricao": "Telefone celular",
11            "produtoPreco": 999.89
12        }, {
13            "produtoId": 89,
14            "produtoDescricao": "Carregador veicular",
15            "produtoPreco": 29.99
16        }]
17    }]
18 }

```

Código 2.3: Estrutura do documento JSON: Cliente, Pedido e Produto (embutidos).

Os bancos de dados baseados em documento são apropriados para problemas que envolvem ambientes variados e lidam com o armazenamento de grandes volumes de dados. Na prática, esse tipo de banco de dados é indicado para armazenar e gerenciar grandes coleções de dados. Além disso, recomenda-se armazenar documentos conceituais como

representações desnormalizadas de uma entidade de banco de dados, além de dados semi-estruturados, que exigem uso de valores nulos [49].

Recomenda-se também o uso de bancos de dados baseados em documento para registro de eventos (*logs*); sistemas de gerenciamento de conteúdo e plataformas de *blog*; análises *web* em tempo real; aplicativos de comércio eletrônico que devem ter esquemas de armazenamento flexíveis.

Alguns exemplos de bancos de dados baseados em documento são MongoDB<sup>33</sup> e CouchDB<sup>34</sup> [52].

#### 2.4.4 Formato GeoJSON

GeoJSON<sup>35</sup> é um formato de intercâmbio de dados geoespaciais baseado na notação JSON, que pode ser utilizado em bancos de dados NoSQL, para armazenamento de dados geográficos. Um objeto GeoJSON pode representar geometrias do tipo ponto (*Point*), linha (*LineString*), polígono (*Polygon*), multiponto (*MultiPoint*), multilinha (*MultiLineString*) e multipolígono (*MultiPolygon*).

A estrutura de dados GeoJSON consiste de um único objeto JSON, sendo este objeto constituído por uma coleção de pares de chave-valor que podem representar uma geometria, um recurso ou coleção de recursos. Posições são fundamentais na construção de geometrias. O campo *coordinates* de um objeto geométrico é composto por uma posição quando representa um ponto; um vetor de posições quando representa uma linha ou multiponto; um vetor de vetores de posições quando representa um polígono ou multilinha; ou um vetor multidimensional de posições (multipolígono). Uma posição é representada por uma série de números, e deve ter, pelo menos, duas coordenadas *x* e *y* (longitude e latitude). Caso seja necessário, é possível representar a posição usando o padrão *x, y, z* (longitude, latitude, altitude), seguindo o sistema de referência de coordenadas geográficas.

O Código 2.4 representa um objeto geométrico do tipo ponto na notação GeoJSON, definindo a posição geográfica do ponto pela longitude (*x*) e latitude (*y*) presente no campo *coordinates*. O Código 2.5 representa o objeto geométrico do tipo linha que contém uma lista de pontos conectados também definidos a partir das coordenadas geográficas de longitude e latitude. O Código 2.6 apresenta a forma geométrica do polígono que também contém uma lista de pontos conectados, mas que fecham um circuito. Nota-se que neste caso, os pontos inicial e final possuem as mesmas coordenadas geográficas de longitude

---

<sup>33</sup><https://www.mongodb.com/>

<sup>34</sup><http://couchdb.apache.org/>

<sup>35</sup><http://geojson.org/>

e latitude. As Figuras 2.9, 2.10 e 2.11 apresentam, respectivamente, as representações gráficas no mapa dos Códigos 2.4, 2.5 e 2.6.

```
1 {  
2   "type": "Point",  
3   "coordinates": [-47.883381, -15.793518]  
4 }
```

Código 2.4: Estrutura do documento GeoJSON: geometria ponto.

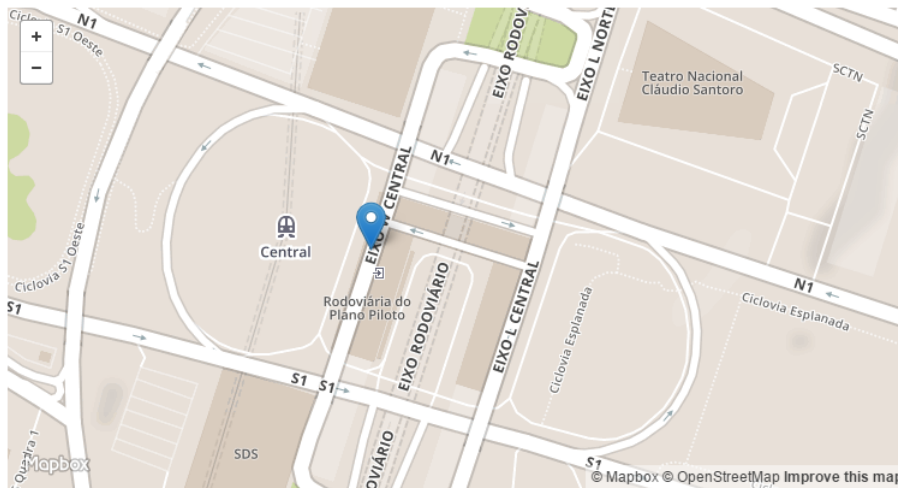


Figura 2.9: Representação gráfica do Código 2.4: geometria ponto usando GeoJSON.

```
1 {  
2   "type": "LineString",  
3   "coordinates": [  
4     [-47.883381, -15.793518],  
5     [-47.883977, -15.795200]  
6   ]  
7 }
```

Código 2.5: Estrutura do documento GeoJSON: geometria linha.

```
1 {  
2   "type": "Polygon",  
3   "coordinates": [  
4     [  
5       [-47.883381, -15.793518],  
6       [-47.883977, -15.795200],  
7       [-47.882745, -15.795869],  
8       [-47.881771, -15.793040],  
9       [-47.883083, -15.792543],  
10      [-47.883381, -15.793518]
```



Figura 2.10: Representação gráfica do Código 2.5: geometria linha usando GeoJSON.

```

11 ]
12 ]
13 }

```

Código 2.6: Estrutura do documento GeoJSON: geometria polígono.



Figura 2.11: Representação gráfica do Código 2.6: geometria polígono usando GeoJSON.

O Código 2.7 apresenta um exemplo de documento escrito com a notação GeoJSON, representando uma geometria de ponto com informações adicionais. Diferentemente dos códigos anteriores, este exemplo possui o campo *properties*, que define propriedades para o objeto; e o campo *geometry*, que define qual forma geométrica será representada pela notação e suas respectivas coordenadas geográficas. O valor *Feature* para o campo *type* define um recurso, onde é possível atribuir propriedades para esse recurso que serão exibidas

em uma caixa de diálogo dentro do mapa como informações adicionais do objeto geométrico. Neste exemplo, representa-se a forma geométrica ponto localizada na Universidade de Brasília. A Figura 2.12 faz a representação gráfica no mapa do Código 2.7.

```
1 {
2   "type": "Feature",
3   "properties": {
4     "instituicao": "Universidade de Brasilia",
5     "cidade": "Brasilia-DF"
6   },
7   "geometry": {
8     "type": "Point",
9     "coordinates": [-47.869041, -15.75851]
10  }
11 }
```

Código 2.7: Estrutura do documento GeoJSON: geometria ponto com informações adicionais.

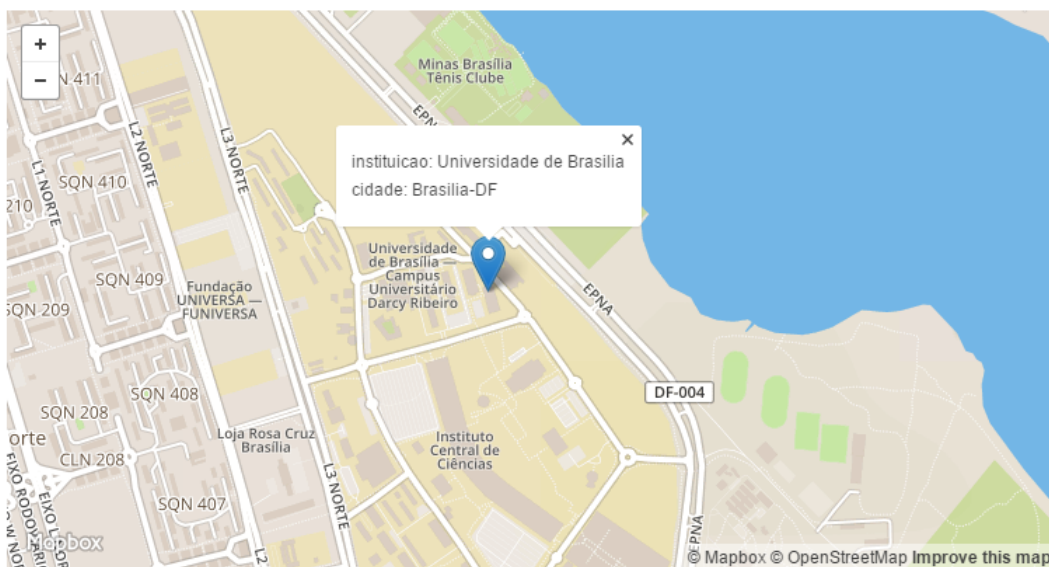


Figura 2.12: Representação gráfica do Código 2.7: geometria ponto com informações adicionais usando GeoJSON.

O objeto GeoJSON pode, opcionalmente, conter um atributo *crs*, cujo valor deve ser um objeto do sistema de referência de coordenadas. O sistema de referência de coordenadas geográficas padrão é o *datum* WGS84, tendo as unidades de longitude e latitude representadas por números decimais<sup>36</sup>.

<sup>36</sup><http://geojson.org/geojson-spec.html>

## 2.5 Trabalhos Relacionados

Na literatura, tem-se artigos que apresentam armazenamento de dados geográficos em bancos de dados NoSQL, assim como artigos sobre SIGV. Os artigos descritos nesta seção foram selecionados por apresentarem estrutura alternativa de armazenamento de dados, na qual atendem requisitos dos SIGV. Além disso, a seleção dos artigos seguiu critérios de relevância ao tema e número de citações como parâmetros de escolha.

### 2.5.1 Armazenamento de Dados Geográficos em Banco de Dados NoSQL

Em Zhang *et al* [62], é possível ver uma abordagem de armazenamento de *big data* espacial através do banco de dados NoSQL baseado em documento MongoDB. O MongoDB é um sistema de gerenciamento de banco de dados que fornece alto desempenho e métodos de armazenamento de dados extensível para aplicações com grandes quantidades de dados. Além disso, possui bom desempenho na consulta de dados. O artigo apresenta a abordagem utilizada para armazenar, consultar e atualizar dados espaciais no MongoDB, utilizando arquivos ESRI *Shapefile* e um *script* desenvolvido em Python. A ideia é ler e analisar arquivos ESRI *Shapefile* (.shp, .shx, e .dbf) para gerar um documento, que será armazenado no MongoDB, contendo o tipo da forma geométrica, os atributos e coordenadas.

Em Baptista *et al* [6], considera-se que os bancos de dados relacionais não são adequados para lidar com grandes quantidades de dados, e cita que os bancos de dados NoSQL baseados em documento CouchDB e MongoDB proveêm suporte a dados espaciais. O artigo propõe uma arquitetura para solucionar problemas de interoperabilidade no armazenamento de dados entre os sistemas de banco de dados relacionais e os NoSQL, através da implementação de serviços OGC, como *Web Map Service* (WMS) e *Web Feature Service* (WFS). Cita-se ainda o trabalho de Miller *et al* [43] que utiliza o CouchDB para armazenar dados espaciais, e interface *mobile* para a recuperação dos dados.

Em Vitolo *et al* [61], sugere-se a utilização de banco de dados multidimensional, como o Rasdaman, para aplicações científicas ambientais e climáticas, que podem utilizar dados *raster* multidimensionais. Recomenda-se também a utilização das variações da XML desenvolvidas para lidar com dados ambientais e geográficos, como *WaterML* e GML (*Geography Markup Language*). Além disso, o artigo diz que a construção de observatórios virtuais do ambiente requer o uso de várias ferramentas para aquisição e análise de dados; e apresentou o projeto EVOp, que é um conjunto de aplicações *web* que utilizam informações ambientais a partir de modelos e ferramentas da comunidade local.

Em Jardak *et al* [32], propõe-se a utilização do banco de dados NoSQL baseado em coluna Hbase, e o *framework* Hadoop para implementação de estruturas de processamento de *big data* espacial; porém, é necessário atentar-se a alguns detalhes no projeto da implementação da arquitetura para se beneficiar do alto grau de paralelismo ativado pelo Hadoop. Em Ma *et al* [37] também recomenda-se a utilização do Hbase para implementação de SIG com dados vetoriais ou com dados de imagem, como em sistemas de sensoriamento remoto.

Em Lizardo *et al* [36], o banco de dados NoSQL baseado em coluna Cassandra é utilizado para armazenar dados geoespaciais em um protótipo de banco de dados espacial, chamado GeoNoSQL. No entanto, até o momento da escrita deste trabalho, o Cassandra não possui mecanismos ou extensões nativas para trabalhar com dados geoespaciais e por isso não é capaz de indexar dados multidimensionais. O trabalho citado utilizou a biblioteca de recuperação de informação Apache Lucene com a extensão geoespacial Lucene *Spatial* para construir o índice espacial. A solução que adotou o Cassandra para o armazenamento de dados pode ser aplicada na construção de SIG que lidam com grandes quantidades de dados, e necessitam de alta escalabilidade e desempenho.

Em Li *et al* [35], há uma ideia similar ao trabalho de Lizardo *et al* [36], e propõe-se a criação de um índice espacial para o banco de dados NoSQL baseado em documento e a grafo OrientDB, na qual pode ser aplicado na distribuição de dados espaciais usando o método GeoHash e satisfazendo uma alta taxa de inserção usando o método de índice árvore B.

Em Pourabbas [50], cita-se também os bancos de dados CouchDB, MongoDB, BigTable e Neo4j como exemplos de sistemas NoSQL que dão suporte aos dados geoespaciais. Neste caso, se fez uma discussão sobre os recursos desejáveis para os sistemas de bancos de dados espaciais e realizou-se uma comparação entre os bancos de dados NoSQL que lidam com dados geográficos. Para efeito de comparação e análise dos bancos de dados NoSQL com recursos geoespaciais utilizou-se a recomendação de recursos desejáveis do padrão ISO SQL/MM. Os recursos analisados foram, por exemplo, suporte a diferentes sistemas de referência de espaço (SRID); métodos de indexação de dados espaciais; tipos de dados vetoriais; funções topológicas, métricas e de conjunto; e os formatos de entrada/saída de dados.

A Tabela 2.1 apresenta um resumo dos objetivos, aplicações e/ou resultados das referências relacionadas. Como é possível observar nos trabalhos anteriores, tem-se interesse acadêmico na área de bancos de dados NoSQL e dados geográficos. A seguir são apresentados alguns trabalhos de arquiteturas de Sistemas de Informação Geográfica Voluntária.

## 2.5.2 Sistemas de Informação Geográfica Voluntária

Apresenta-se aplicações de Sistemas de Informação Geográfica Voluntária a fim de ter uma visão geral da organização desse tipo de SIG. A partir do intuito de propor uma arquitetura de armazenamento de dados específica para SIGV é necessário conhecer implementações que podem nortear a proposta a ser apresentada. Neste caso, todas as aplicações preveem a utilização de dados oriundos de usuários voluntários.

Em Sheppard [54], propõe-se um *framework*, denominado “wq”, com elevado grau de reutilização para aplicações de Informação Geográfica Voluntária (IGV), utilizando padrões e componentes de código aberto. Essa proposta, incentiva a utilização de códigos genéricos que podem ser executados tanto em plataformas *web* quanto *mobile*, e é constituída por três bibliotecas que permitem a coleta e armazenamento de dados, além da utilização do SIGV.

A estrutura do *framework* proposto em Sheppard [54] utiliza uma abordagem modular, e adota HTML5, CSS3 e a biblioteca *javascript* JQuery na interface de coleta de dados. Em sua biblioteca de armazenamento, utiliza-se o *framework* Django baseado em Python, que abstrai a tarefa de geração de consultas de banco de dados (modelo objeto-relacional). O Django pode ser aplicado com vários sistemas de bancos de dados, incluindo PostgreSQL/PostGIS. Para interação com os clientes do SIGV a biblioteca de armazenamento utiliza a API REST, Django REST *Framework*. A biblioteca de utilização abstrai o processo de transformação de dados e arquivos de entrada/saída.

Em Miranda *et al* [44], é proposta uma arquitetura para sistemas de informação que recebe informações geográficas voluntárias em uma Infraestrutura de Dados Espaciais (IDE) em nível municipal. O sistema possui dois módulos que fornecem funcionalidades para gestão da informação colaborativa e outro para receber contribuições. A estrutura da arquitetura possui 3 camadas: Dados, Negócios e Apresentação. Padrões e especificações propostas pela OGC foram adotados para simplificar a interoperabilidade de conteúdos espaciais através de *web services*. Todos são softwares livres: I3Geo, MapServer, PostgreSQL e Geonetwork. Acesso e descrição de metadados são providos pelo Geonetwork. O repositório de metadados é mantido pelo PostgreSQL. MapServer garante a implementação do *web services* OGC, enquanto a visualização e análise dos dados espaciais são providos pelo visualizador do I3Geo.

Em Davis Júnior *et al* [18], propõe-se um *framework* para ser usado na criação de várias aplicações de informação geográfica voluntária. A proposta inclui elementos necessários para personalizar a coleta de informação, permitindo uma estrutura e interface unificada para plataformas *web* e *mobile*. A ideia desse *framework* é encapsular uma estrutura básica de SIGV em blocos extensíveis que podem ser reutilizados em novos projetos.



A arquitetura geral do *framework* proposto em Davis Júnior *et al* [18] é dividida em três camadas: Camada de apresentação, que utiliza o padrão *Model-View-Controller (MVC)*, e é responsável pela coleta dos dados nos ambientes *web* e *mobile*; Camada de negócios, que é responsável pela definição dos protocolos de rede, pelos serviços de inserção e recuperação de dados, e pelos *drivers* de acesso ao SGBD; e Camada de dados, responsável por padronizar o esquema conceitual do banco de dados utilizando um esquema genérico com três grupos de objetos – usuários, contribuições e avaliação de contribuições.

O trabalho de Davis Júnior *et al* [18] utilizou o *software* Strepitus, que objetiva estimar o nível de ruído na região onde está localizado o usuário, para validar a proposta do *framework*. A camada de apresentação implementa interfaces para iOS, Android e *web*, sendo utilizada as linguagens PHP e *javascript*, além das bibliotecas OpenLayers, ExtJS e GeoExt. A camada de negócios utiliza a linguagem PHP, HTTP (protocolo de transmissão de mensagens), JSON (formato das mensagens), e o *driver* de acesso ao banco de dados PostgreSQL (PostGIS) para PHP. Ainda na camada de negócios, o GeoServer é utilizado para prover os dados a serem exibidos pela aplicação, além de possibilitar a conexão a diversas fontes de dados e a publicação como *web services* OGC. A camada de dados adota o banco de dados PostgreSQL com a extensão espacial PostGIS para armazenamento dos dados da aplicação.

Em Camargos *et al* [11], propõe-se uma arquitetura de coleta de opinião sobre serviços públicos em um Sistema de Informação Geográfica Móvel com Participação Popular (SIGPP). A arquitetura baseia-se em um banco de dados central responsável por armazenar informações sobre estabelecimentos públicos e suas respectivas avaliações, realizadas por usuários voluntários. Essa arquitetura é organizada pela camada de apresentação, que inclui a uma interface móvel e uma interface *web*; pela camada de comunicação de dados, que é responsável por fornecer os dados necessários para o funcionamento das interfaces; pela camada de dados, que apresenta o modelo conceitual do banco de dados; e pelo Sistema Gerenciador de Banco de Dados que é responsável pela consistência e manipulação dos dados da camada de dados.

Para validação de sua arquitetura, Camargos *et al* [11] desenvolveu o aplicativo Android *ConsultaOpinião* para implementar a interface móvel da arquitetura, assim como implementou a interface *web* utilizando a linguagem PHP. Para implementação da camada de comunicação de dados e da camada de dados desenvolveu-se algoritmos para validação de usuários, obtenção de locais (estabelecimentos públicos) cadastrados, verificação da localização do usuário, obtenção das perguntas dos questionários, entre outras funções importantes para a modelagem do SIGPP. O Sistema Gerenciador de Banco de Dados PostgreSQL com a extensão espacial PostGIS foi adotado pela arquitetura para lidar com o gerenciamento dos dados da aplicação.

A arquitetura de Camargos *et al* [11] é explorada no capítulo 4 deste trabalho de dissertação, pois utilizou-se o aplicativo *ConsultaOpinião* em uma das provas de conceito da arquitetura proposta.

A Tabela 2.2 apresenta um resumo dos objetivos, banco de dados utilizado, aplicações e/ou resultados dos Sistemas de Informação Geográfica Voluntária relacionados.

Tabela 2.1: Trabalhos Relacionados: Armazenamento de dados geográficos em banco de dados NoSQL.

<b>Autores</b>	<b>Ano</b>	<b>Objetivo</b>	<b>Tipo de banco de dados</b>	<b>Aplicação/Resultado</b>
Zhang <i>et al</i> [62]	2014	Apresentar a abordagem utilizada para armazenar, consultar e atualizar dados espaciais no MongoDB, utilizando arquivos ESRI <i>Shapefile</i> e um <i>script</i> desenvolvido em Python.	Baseado em Documento	Ler e analisar arquivos ESRI <i>Shapefile</i> para gerar um documento, que será armazenado no MongoDB, contendo o tipo da forma geométrica, os atributos e coordenadas.
Baptista <i>et al</i> [6]	2011	Propor uma arquitetura para solucionar problemas de interoperabilidade no armazenamento de dados entre os sistemas de banco de dados relacionais e os NoSQL, através da implementação de serviços OGC, como WMS e WFS.	Baseado em Documento	Apresenta os bancos de dados NoSQL CouchDB e MongoDB como adequados para lidar com grandes quantidades de dados e armazenar dados espaciais.
Vitolo <i>et al</i> [61]	2015	Apresentar os desafios para o armazenamento e gerenciamento de uma grande quantidade de dados ambientais, e propor tecnologias que possam realizar essas tarefas de maneira mais eficiente.	Banco de Dados multidimensional	Sugere-se a utilização de banco de dados multidimensional para aplicações científicas ambientais e climáticas, recomendando também a utilização de arquivos no padrão <i>WaterML</i> e <i>GML</i> .
Jardak <i>et al</i> [32]	2014	Propor a utilização do banco de dados NoSQL Hbase, e o <i>framework</i> Hadoop para implementação de estruturas de processamento de <i>big data</i> espacial.	Baseado em Coluna	Armazenamento de dados vetoriais ou de imagem em SIGs, utilizando o Hbase.
Lizardo <i>et al</i> [36]	2014	Apresentar um protótipo de banco de dados espacial, chamado GeoNoSQL, que utiliza o banco de dados NoSQL Cassandra para o armazenamento de dados.	Baseado em Coluna	Construção de SIGs que lidam com grandes quantidades de dados, e necessitam de alta escalabilidade e desempenho.
Li <i>et al</i> [35]	2013	Propor a criação de um índice espacial para o banco de dados NoSQL OrientDB.	Baseado em Documento e a Grafo	Distribuir dados espaciais usando o método GeoHash e satisfazendo uma alta taxa de inserção usando o método de índice árvore B.
Pourabbas [50]	2014	Apresentar tendências e tecnologias dos Sistemas de Informação Geográfica.	Bancos de dados NoSQL e relacionais	Apresenta o CouchDB, MongoDB, BigTable e Neo4j como exemplos de sistemas NoSQL que dão suporte aos dados geoespaciais, e faz uma discussão sobre os recursos desejáveis para os sistemas de bancos de dados espaciais.

Tabela 2.2: Trabalhos Relacionados: Sistemas de Informação Geográfica Voluntária.

<b>Autores</b>	<b>Ano</b>	<b>Objetivo</b>	<b>Banco de Dados utilizado</b>	<b>Aplicação/Resultado</b>
Sheppard [54]	2012	Propor um <i>framework</i> com elevado grau de reutilização para aplicações que utilizam dados de usuários voluntários, adotando padrões e componentes de código aberto.	PostgreSQL / PostGIS	Implementação de SIGV a partir de um <i>framework</i> .
Miranda <i>et al</i> [44]	2011	Propor uma arquitetura para sistemas de informação que recebe informações geográficas voluntárias em uma IDE em nível municipal.	PostgreSQL / PostGIS	Adaptação de uma arquitetura de IDE para sistemas de informação geográfica voluntária, apresentando a aplicação Viçosa Digital que utiliza dados colaborativos para auxiliar no planejamento estratégico municipal.
Davis Júnior <i>et al</i> [18]	2013	Propor um <i>framework</i> para ser usado na criação de várias aplicações de informação geográfica voluntária, através de uma estrutura básica organizada em blocos extensíveis que podem ser reutilizados em novos projetos.	PostgreSQL / PostGIS	Modelo de implementação para aplicações de SIGV, que apresentou o aplicativo Strepitus para validar sua proposta.
Camargos <i>et al</i> [11]	2015	Propor uma arquitetura de coleta de opinião sobre serviços públicos em um SIGPP, que baseia-se em um banco de dados central responsável por armazenar informações.	PostgreSQL / PostGIS	Aplicativo Consulta Opinião que possibilita a avaliação de estabelecimentos públicos a partir de dados de usuários voluntários.

# Capítulo 3

## Arquitetura Proposta

Neste capítulo propõe-se uma arquitetura de armazenamento de dados para Sistemas de Informação Geográfica Voluntária (SIGV), utilizando banco de dados NoSQL baseado em documento, que contemple requisitos como escalabilidade e heterogeneidade de dados. A Seção 3.1 descreve o problema, como vem sendo tratado e a utilidade de sua resolução. A Seção 3.2 descreve uma arquitetura abstrata de SIGV e seus respectivos itens. A Seção 3.3 apresenta a arquitetura de armazenamento de dados, e descreve seus módulos e processos internos. A Seção 3.4 descreve como a arquitetura de armazenamento proposta foi implementada.

### 3.1 Descrição do Problema

Os Sistemas de Informação Geográfica Voluntária (SIGV), possuem características que necessitam de estratégias adequadas para o armazenamento eficiente de seus dados. É necessário lidar com questões que envolvem o armazenamento de uma grande quantidade de dados e o acesso simultâneo de múltiplos usuários, que podem ocasionar muitas operações de leitura e gravação.

Outro problema inerente aos SIGVs, é a diversidade de formatos que podem ser armazenados nas aplicações. O sistema de banco de dados adotado pelo SIGV deve possibilitar flexibilidade em relação aos tipos de dados permitidos para armazenamento, principalmente considerando que poderão ser armazenados dados geográficos e arquivos de imagem, áudio e vídeo.

Diante desses desafios, é possível sugerir que os SIGVs necessitam de uma arquitetura de armazenamento de dados robusta, que adota tecnologias alternativas, como os bancos de dados NoSQL, para o armazenamento eficiente de seus dados. Atualmente, o armazenamento de dados geográficos em bancos de dados NoSQL, vem sendo explorado por pesquisadores com o objetivo de se criar uma arquitetura distribuída em banco de

dados, e que suporte o armazenamento e processamento de uma grande quantidade de informações.

Não se encontrou na literatura uma proposta específica de arquitetura de armazenamento de dados para Sistemas de Informação Geográfica Voluntária (SIGV), que solucione explicitamente os problemas citados. Assim, espera-se que essa arquitetura seja útil para propor formas alternativas de armazenamento de dados para aplicações que possuem características dos SIGVs, e que busque contemplar requisitos de escalabilidade e heterogeneidade de dados. A escalabilidade aqui citada, refere-se a possibilidade de aumentar a capacidade de processamento e armazenamento de dados, suportando um aumento da carga de trabalho. Por sua vez, a heterogeneidade (variedade) dos dados considerada neste trabalho, é a forma que um sistema de armazenamento lida com dados de variados formatos.

## 3.2 Proposta de Arquitetura Abstrata para SIGV

Usualmente, uma arquitetura para SIGV inclui a definição de políticas de acesso de usuários; de verificação e validação de informações voluntárias; padrões de interface para visualização e entrada de dados; metadados; e tecnologias que são necessárias para implementação da infraestrutura do sistema de informação [44, 21, 54, 4, 18].

A arquitetura proposta neste trabalho é focada no armazenamento de dados de diversos formatos em SIGV utilizando banco de dados NoSQL e, portanto, não tem a preocupação de apresentar detalhadamente todos os itens presentes em uma arquitetura típica de SIGV. No entanto, propõe-se inicialmente uma arquitetura abstrata para possibilitar uma melhor compreensão da organização dos SIGV.

A Figura 3.1 exibe uma arquitetura abstrata de um SIGV, que se apresenta em uma organização em três camadas: Apresentação, Negócios e Persistência de Dados. As camadas de Apresentação e Negócios compõem a estrutura da aplicação cliente, ou seja, o SIGV que fará a integração com o sistema de banco de dados NoSQL presente na camada de Persistência de Dados.

Os itens da arquitetura abstrata de um SIGV, proposta neste trabalho, são detalhados a seguir:

- Camada de Apresentação: nesta camada a aplicação deve disponibilizar uma interface intuitiva e de fácil utilização para visualização e entrada de dados, suportada preferencialmente em dispositivos móveis e *desktop*.
- Camada de Negócios: esta camada realiza funções de processamento; interação com o serviço de distribuição de mapas; e comunicação com a camada de persistência

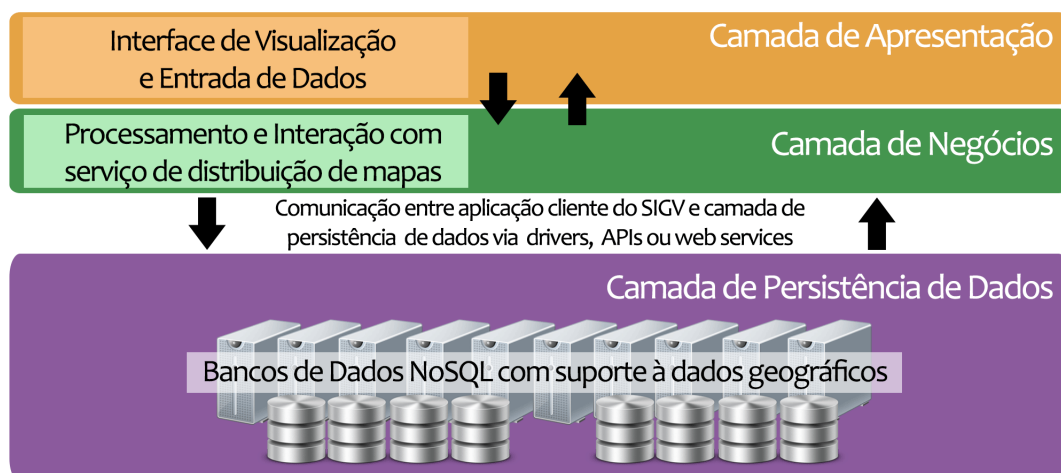


Figura 3.1: Arquitetura abstrata de um SIGV utilizando Banco de Dados NoSQL.

de dados. A comunicação da camada de negócios com a camada de persistência de dados ocorre através de *drivers*, específicos para linguagem de programação utilizada na implementação do SIGV, via API (*Application Programming Interface*) ou de *web services* [50].

- Camada de Persistência de Dados: esta camada é responsável por armazenar os dados do SIGV em um sistema de banco de dados NoSQL com suporte a dados geográficos, e que atenda aos requisitos de escalabilidade e heterogeneidade de dados.

Na Seção 3.3 é feita a apresentação da arquitetura de armazenamento de dados para SIGV proposta.

### 3.3 Proposta de Arquitetura de Armazenamento de Dados para SIGV

A arquitetura proposta tem o intuito de armazenar todos os dados da aplicação em um Sistema Gerenciador de Banco de Dados, sejam eles dados convencionais (dados alfanuméricos) ou dados não convencionais, como dados geográficos, dados em formato binário, entre outros.

Pelo fato dos sistemas de bancos de dados NoSQL serem escaláveis, fornecerem suporte a variados tipos de dados, e terem condições de trabalhar com dados estruturados, semiestruturados ou não estruturados, pode-se inferir que os bancos de dados NoSQL sejam uma boa alternativa para o armazenamento de dados em SIGV, que demandam o armazenamento de diversos tipos de dados e podem ser utilizados por muitos usuários.

Esta arquitetura de armazenamento de dados para SIGV adota bancos de dados NoSQL baseados em documento, pois esse tipo de banco de dados oferece maior robustez no armazenamento de dados geográficos, suportando nativamente o padrão para intercâmbio de dados geoespaciais GeoJSON; e ainda oferece suporte ao armazenamento de uma grande variedade de dados, podendo carregar qualquer tipo de documento sem a necessidade de conhecer previamente a estrutura do arquivo. Os bancos de dados baseados em documento também fornecem consulta sobre todos os dados armazenados no banco de dados, sendo um recurso interessante para qualquer aplicação.

Adotando-se a técnica de banco de dados distribuído, é possível se beneficiar da arquitetura de armazenamento e gerenciamento de dados distribuídos. Assim, os sistemas de banco de dados NoSQL podem oferecer maior capacidade de armazenamento, maior concorrência de leitura e escrita, bem como melhor disponibilidade e tolerância a falhas [37].

A Figura 3.2 apresenta a arquitetura distribuída de armazenamento de dados proposta para SIGV utilizando banco de dados NoSQL baseado em documento. Essa arquitetura segue a organização típica em três camadas – apresentação, negócios e persistência de dados – conforme apresentado na Seção 3.2.



Figura 3.2: Arquitetura de Armazenamento de Dados para SIGV proposta.



A comunicação entre a aplicação cliente do SIGV, presente nas camadas de apresentação e negócios, e a camada de persistência de dados, é realizada por intermédio de requisições HTTP para os *RESTful web services* (Figura 3.2). Os *web services*, presentes na camada de persistência de dados, por sua vez, ficam responsáveis por tratar as requisições das aplicações e estabelecer comunicação com o banco de dados NoSQL, com suporte a dados geográficos.

Os *RESTful web services* são baseados no padrão da arquitetura REST (*Representational State Transfer*) e utilizam o protocolo HTTP para comunicação de dados. Esses *web services* fornecem acesso a recursos, que são identificados por um *Uniform Resource Identifier (URI)*, e podem utilizar os métodos *GET*, *POST*, *PUT*, *DELETE* e *OPTIONS* do protocolo HTTP.

Nesta proposta de arquitetura, as requisições HTTP feitas pela aplicação devem utilizar os métodos *GET* e *POST* para recuperar e armazenar dados a partir dos *web services*. Os *web services*, por sua vez, devem retornar uma mensagem em formato JSON dando uma resposta a requisição do usuário da aplicação. Cada serviço (recurso) dos *web services* correspondem a uma operação de leitura ou escrita de dados no banco de dados NoSQL. Assim, esses *web services* devem ser implementados em uma linguagem que suporte o *driver* específico do banco de dados adotado na camada de persistência de dados.

Optou-se por utilizar os *web services* na intermediação da comunicação da aplicação cliente do SIGV e a camada de persistência de dados, pois a tecnologia proporciona a adoção de uma coleção de padrões abertos, como JSON ou XML, para troca de dados entre sistemas e aplicações, que podem ser escritas em diversas linguagens e que executam sob diferentes plataformas, tornando a camada de persistência de dados interoperável. Além disso, a partir da implementação dos *web services*, é possível trabalhar com interfaces móveis e *desktop* sem a necessidade de reprogramar os serviços disponíveis que realizam diretamente o armazenamento e recuperação de dados no banco de dados.

Os *web services* possuem, além dos recursos disponíveis para utilização das aplicações clientes, o módulo de integração e conversão de dados; e o módulo de funções de consulta e análise espacial, que podem ser acessados nas operações de recuperação e armazenamento de dados. Os módulos citados possuem as seguintes funções:

- Módulo de Integração e Conversão de Dados: este módulo é responsável pela organização sintática dos dados em uma notação específica para posterior armazenamento dos dados heterogêneos, incluindo dados geográficos, no sistema de banco de dados NoSQL.
- Módulo de Funções de Consulta e Análise Espacial: este módulo é responsável por implementar funções topológicas; análise e funções métricas; e funções de conjuntos,

adicionais ou complementares às já existentes no banco de dados NoSQL adotado na arquitetura.

O processo de armazenamento de dados em SIGV utilizando banco de dados NoSQL baseado em documento, conforme apresentado na Figura 3.3, compreende as seguintes atividades:

- Realizar requisição HTTP: a aplicação cliente do SIGV faz a requisição de armazenamento dos dados para o *web service*, enviando os dados informados pelo usuário voluntário através do método de envio POST do protocolo HTTP.
- Tratar requisição HTTP/Receber os dados: o *web service* recebe os dados enviados pela aplicação e encaminha-os para a atividade responsável pela análise do formato dos dados recebidos.
- Analisar o Formato dos Dados: o *web service* faz a verificação dos dados recebidos. Caso os dados estejam em formato binário, procede-se a comunicação com o banco de dados para posterior gravação dos dados no banco. Caso contrário, os dados convencionais ou geográficos são encaminhados para a atividade de integração e estruturação de dados.
- Integrar e Estruturar os Dados: o *web service* faz a estruturação dos dados recebidos nas notações JSON/GeoJSON, a partir do Módulo de Integração e Conversão de Dados, para armazenamento dos dados no banco de dados NoSQL.
- Realizar comunicação com o Banco de Dados: o *web service*, através do *driver* específico para o banco de dados adotado na camada de persistência de dados, faz a comunicação com o servidor de banco de dados NoSQL.
- Enviar Dados para o Banco de Dados: o *web service* envia os dados para serem armazenados no banco de dados NoSQL, estruturados no formato JSON/GeoJSON no caso de dados convencionais ou geográficos, e sem a necessidade de conversão, no caso de dados no formato binário.
- Receber/Armazenar Dados: o banco de dados NoSQL recebe os dados enviados pelo *web service* e realiza o armazenamento físico dos dados inseridos pelo usuário do SIGV.
- Confirmar Armazenamento de Dados: o banco de dados NoSQL faz a confirmação do armazenamento dos dados, enviando o status da operação.
- Receber/Enviar Confirmação de Armazenamento de Dados: o *web service* recebe e verifica o status da operação de gravação de dados realizada pelo banco de dados,

e encaminha uma mensagem no formato JSON para a aplicação cliente, atestando ou não a gravação.

- Receber resposta HTTP: a aplicação cliente do SIGV recebe o retorno da requisição HTTP através de um documento JSON, que confirma ou não a gravação dos dados enviados.

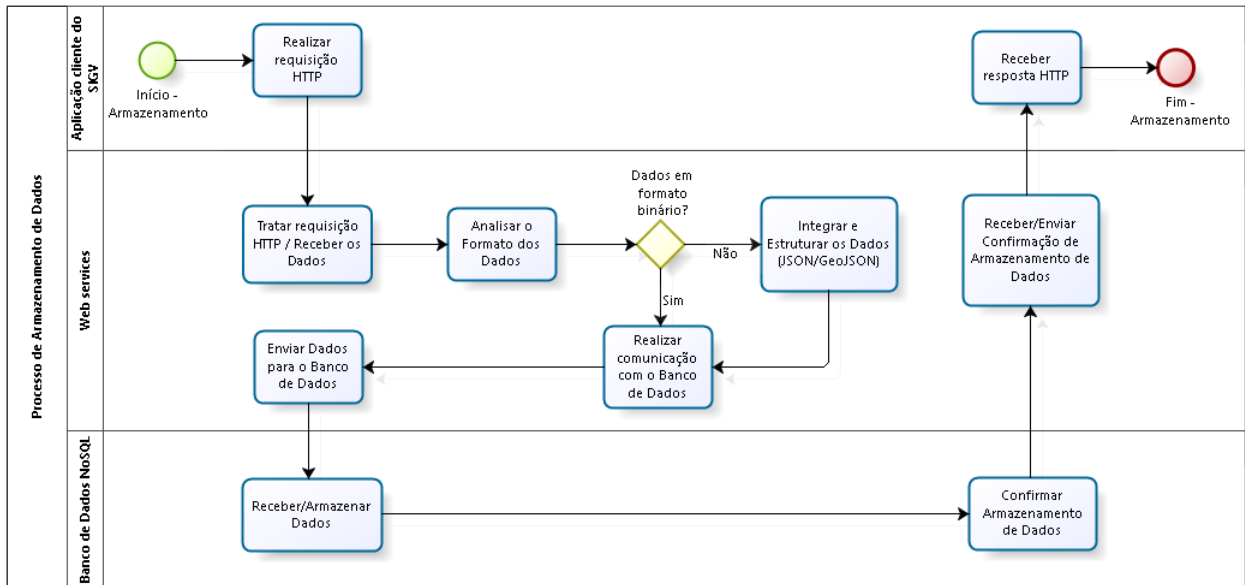


Figura 3.3: Processo de armazenamento de dados da arquitetura proposta.

O processo de armazenamento de dados em SIGV pode lidar com o armazenamento de diversos tipos de arquivos binários, como imagem, áudio e vídeo, para documentar um relato ou contribuição de um determinado usuário. Neste caso, a arquitetura deve estar preparada para receber e analisar o formato desses tipos de arquivos; além dos dados convencionais e geográficos que são utilizados com frequência.

Para o armazenamento de arquivos binários, aproveita-se da flexibilidade dos bancos de dados baseados em documento para realizar a gravação dos dados sem a necessidade de conhecer previamente a estrutura do documento a ser gerenciado pelo SGBD. Assim, a gravação do arquivo binário ocorre diretamente nos sistemas de arquivos do banco de dados, tendo cada tecnologia de banco de dados NoSQL baseado em documento suas particularidades e características.

Em alguns casos, a aplicação do SIGV pode encaminhar dados do GPS do dispositivo cliente para realizar alguma consulta espacial no banco de dados, podendo utilizar o Módulo de Funções de Consulta e Análise Espacial dos *web services* para auxiliar nas operações de recuperação de informações e consultas espaciais. Neste caso, é necessário

realizar a conversão desses dados no formato GeoJSON, que foi adotado por esta arquitetura no armazenamento e recuperação de dados geográficos, pois trata-se de um formato aberto, bastante utilizado e bem adaptado às necessidades de armazenamento de dados em um SIGV, que demanda flexibilidade e variedade de formatos.

A indisponibilidade de um banco de dados pode causar transtornos a seus usuários e, dependendo da aplicação, gerar prejuízos. Os sistemas de banco de dados NoSQL geralmente possuem arquiteturas de replicação de dados que proporcionam uma maior disponibilidade de seu serviço. A arquitetura de armazenamento de dados proposta, recomenda a replicação dos dados a fim de possibilitar um nível de tolerância a falhas, mantendo assim o sistema em funcionamento mesmo que aconteça algum incidente com algum dos servidores de banco de dados disponíveis. A arquitetura de replicação deve ser definida de acordo com o banco de dados NoSQL escolhido na implementação, sendo *master-slave*, *master-master* e *peer-to-peer* as arquiteturas disponíveis, como descritas no capítulo anterior.

Para aplicações que exigem um banco de dados mais robusto, devido a quantidade de dados a ser armazenada ou do acesso de múltiplos usuários simultâneos, recomenda-se a adoção da técnica de *sharding* (fragmentação) para realizar o balanceamento da carga de trabalho entre os servidores de banco de dados NoSQL, disponíveis no *cluster*. A ideia do *sharding* é proporcionar escalabilidade horizontal para obter a vantagem de melhorar o desempenho do banco de dados mediante a redução do número de dados em cada servidor. Nesta técnica, os dados podem ser divididos e distribuídos em múltiplos servidores.

### 3.4 Implementação da Arquitetura Proposta

A implementação da arquitetura de armazenamento de dados proposta pode ser realizada de três maneiras: utilizando um único servidor de banco de dados (modo *standalone*), utilizando um conjunto de servidores de banco de dados replicados, ou utilizando um banco de dados dividido em fragmentos (*sharding*).

A Figura 3.4 mostra o esquema de implementação da arquitetura utilizando um único servidor de banco de dados. Nesta alternativa de implementação, não é possível oferecer tolerância a falhas, pois todos os dados da aplicação ficam centralizados em uma única máquina. Desta forma, a ocorrência de uma falha no servidor de banco de dados acarreta interrupção no funcionamento do SIGV.

Em todas as alternativas de implementação, a comunicação do SIGV com os *web services* ocorre através de requisições HTTP, utilizando os métodos GET e POST, que, na maioria dos casos, retornam mensagens no formato JSON. Os *web services* possuem



Figura 3.4: Esquema da implementação da arquitetura utilizando um único servidor de BD.

vários serviços que podem ser utilizados pelos SIGV e, geralmente, fazem a manipulação de informações no banco de dados.

A comunicação entre os *web services* e o banco de dados ocorre através de *drivers* específicos para a linguagem de programação utilizada na implementação dos *web services*. Na implementação dos *web services* utilizou-se a linguagem PHP 5.6, o *framework* PHP CodeIgniter 3, e o pacote CodeIgniter REST Server, para implementação do servidor RESTful. Neste caso, a comunicação dos *web services* com o banco de dados foi estabelecida da seguinte maneira:

- Utilizando a extensão `php_pgdsql.dll`<sup>1</sup> (padrão do PHP) para comunicação com o banco de dados PostgreSQL.
- Utilizando a extensão `php_mongo.dll`<sup>2</sup> instalada no servidor e configurada para iniciar em conjunto com a linguagem PHP para conexão com o banco de dados MongoDB.
- Utilizando *sockets* para conexão com o banco de dados CouchDB na porta 5984.

A Figura 3.5 apresenta a implementação da arquitetura utilizando um conjunto de servidores de banco de dados replicados. Essa configuração provê tolerância a falhas a partir da replicação dos dados em diversos servidores presentes na camada de persistência de dados. A replicação aumenta a disponibilidade dos dados da aplicação, já que em uma possível falha de um dos servidores de banco de dados, o SIGV pode utilizar um outro servidor em funcionamento.

Uma implementação desta alternativa de armazenamento de dados foi realizada utilizando um servidor de banco de dados MongoDB configurado com 3 máquinas, sendo uma delas o servidor de banco de dados *master* (principal), responsável pelas operações de leitura/escrita dos dados. Os servidores de banco de dados *slave* (secundários) são réplicas do servidor de banco de dados *master*, mas por padrão, somente armazenam a

<sup>1</sup>[http://php.net/manual/pt\\_BR/pgsql.installation.php](http://php.net/manual/pt_BR/pgsql.installation.php)

<sup>2</sup><https://pecl.php.net/package/mongo>

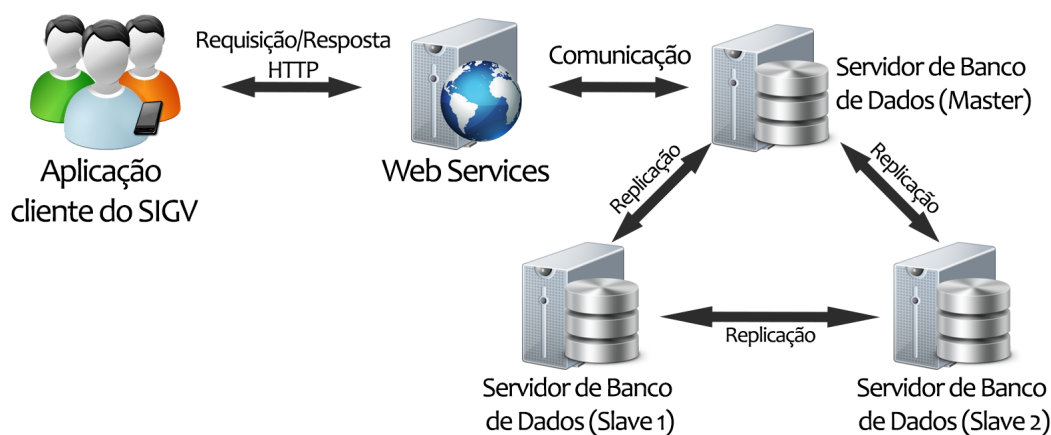


Figura 3.5: Esquema da implementação da arquitetura utilizando servidor de banco de dados replicado.

cópia dos dados do servidor principal. Caso seja necessário, é possível configurar que os servidores secundários no MongoDB sirvam para leitura de dados. No entanto, não se recomenda habilitar essa configuração, pois a replicação dos dados no MongoDB é realizada de forma assíncrona, podendo-se obter cópias diferentes do banco de dados nos nós secundários durante segundos que precedem o término da sincronização.

A configuração de um conjunto de servidores de banco de dados replicados no MongoDB segue os seguintes passos:

1. Criação do conjunto de réplicas.
2. Configuração do ambiente do banco de dados.
3. Inicialização do banco de dados.

A criação do conjunto de réplicas no MongoDB ocorre através da execução do processo *mongod* em todos os servidores participantes do *cluster*. A configuração do ambiente do banco de dados replicado é realizada através do Código 3.1 a seguir. Para adicionar novas réplicas ao conjunto, basta adicionar o IP e a porta de comunicação do servidor de banco de dados da nova máquina do *cluster* e reconfigurar o ambiente replicado.

---

```

1  cfg = {
2      "_id" : "rs0",
3      "version" : 1,
4      "members" : [
5          {
6              "_id" : 0,
7              "host" : "192.168.163.119:27017"
8          },

```

```

9      {
10         "_id" : 1,
11         "host" : "192.168.163.101:27017"
12     },
13     {
14         "_id" : 2,
15         "host" : "192.168.163.115:27017"
16     }
17 ]
18 }

```

---

Código 3.1: Configuração do conjunto de réplicas no MongoDB replicado.

O campo `_id` representa o nome do conjunto de réplicas; o campo `version` apresenta a versão da configuração, que pode ser sequencial; e o campo `members` lista todos os servidores participantes do conjunto de réplicas, atribuindo-lhes um identificador e armazenando seus endereços IPs e porta de comunicação do banco de dados MongoDB.

A Figura 3.6 apresenta a implementação da arquitetura utilizando um *cluster* fragmentado (*sharded cluster*). Essa alternativa de implementação objetiva obter maior robustez e disponibilidade para o sistema de armazenamento de dados da aplicação, visto que os dados são distribuídos em várias máquinas do *cluster* e pode-se realizar um balanceamento da carga de trabalho do servidor de banco de dados.

Realizou-se a configuração de um *cluster* fragmentado utilizando o banco de dados MongoDB trabalhando com 10 máquinas, sendo: 1 máquina (*router*) executando uma instância *mongos*; 3 máquinas para servidores de configuração; e 6 máquinas para servidores de banco de dados. A máquina *router* tem a função de direcionar as requisições de consulta para o servidor de banco de dados que contém os dados requisitados. Os servidores de configuração mantêm os metadados para o *cluster* fragmentado. Esses metadados dizem respeito à organização de todos os dados dentro do *cluster*, incluindo a lista de blocos (porções de dados – *chunks*) presentes em cada fragmento (*shard*), e os intervalos que definem os blocos. Nesta implementação, os servidores de banco de dados foram divididos em 2 fragmentos (A e B), sendo cada fragmento um conjunto de réplicas, ou seja, servidores de banco de dados replicados.

A configuração de um *cluster* fragmentado no MongoDB segue os seguintes passos:

1. Criação do conjunto de réplicas para os servidores de configuração.
2. Inicialização das instâncias do processo *mongos*.
3. Inclusão de fragmentos ao *cluster*.
4. Habilitação da fragmentação para o banco de dados.

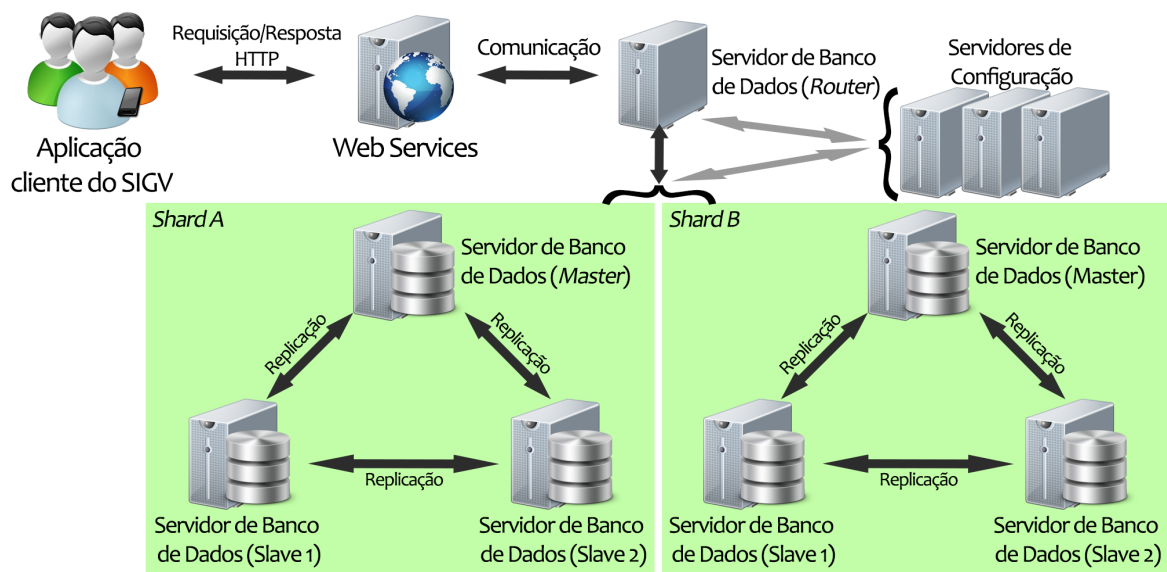


Figura 3.6: Esquema da implementação da arquitetura utilizando *sharded cluster*.

##### 5. Especificação das coleções que serão fragmentadas.

A criação do conjunto de réplicas para os servidores de configuração do *cluster* fragmentado no MongoDB, ocorre através da execução do processo *mongod* em cada servidor de configuração. A execução desse processo deve acompanhar o parâmetro *configsvr* para definir que trata-se de um servidor de configuração de um *cluster*.

A inicialização das instâncias do processo *mongos* cria um *router*, ou seja, um processo roteador, que direciona as requisições das aplicações clientes do banco de dados para o servidor que contém os dados solicitados.

Para adicionar fragmentos ao *cluster*, é necessário conectar-se a uma instância *mongos* e adicionar uma máquina (membro) de um conjunto de servidores replicados ao *cluster*. Para isso, deve-se criar inicialmente um conjunto de réplicas do servidor de banco de dados, de maneira semelhante à configuração dos servidores de banco de dados replicados, e em seguida, adicionar um membro do conjunto de servidores de banco de dados ao *cluster*. Adicionando os fragmentos ao *cluster* é possível especificar os bancos de dados e as coleções que serão fragmentadas.

O capítulo a seguir apresenta a validação da arquitetura proposta através do desenvolvimento de duas provas de conceito, que realizam testes sobre essas alternativas de implementação da arquitetura.



# Capítulo 4

## Provas de Conceito

Neste capítulo são apresentadas duas provas de conceito da arquitetura especificada no capítulo anterior. As provas de conceito fazem a demonstração da aplicação prática dos conceitos e tecnologias apresentadas na arquitetura proposta, avaliando os resultados do desempenho das implementações da arquitetura utilizando bancos de dados NoSQL, e comparando-os com os resultados de uma implementação que utiliza banco de dados tradicional. A primeira prova, Seção 4.1, apresenta o aplicativo *Consulta Opinião* [11]; e a segunda prova, Seção 4.2, apresenta o aplicativo *Comune* [19], ambos desenvolvidos durante o trabalho de conclusão do curso de Ciência da Computação da Universidade de Brasília, por alunos orientados pela Profa. Dra. Maristela Terto de Holanda.

### 4.1 Prova de Conceito 1: Consulta Opinião

Para uma validação inicial da arquitetura de armazenamento de dados, utilizou-se um Sistema de Informação Geográfica Voluntária, o projeto “Consulta Opinião” [11], que tem o objetivo de obter dados sobre a opinião dos usuários dos serviços públicos e apresentá-los ao gestor desses serviços.

A aplicação possui uma interface móvel (aplicativo Android para *smartphones*) voltada para utilização dos usuários dos serviços públicos, que realizam a avaliação dos estabelecimentos cadastrados no sistema; e uma interface *web* (desenvolvida em PHP), direcionada aos gestores dos serviços públicos, que obtém uma visualização gráfica e dinâmica, através do mapa, das avaliações realizadas por usuários voluntários.

O projeto “Consulta Opinião” utiliza o SGBD Objeto-Relacional PostgreSQL, mas foi adaptado para comunicar-se com o banco de dados NoSQL baseado em documento MongoDB, buscando utilizar uma forma alternativa de armazenamento de dados. Como o “Consulta Opinião” foi desenvolvido de forma modular, basicamente, foi necessário

modificar apenas a camada de persistência de dados do aplicativo para o Banco de Dados MongoDB.

A Figura 4.1 exibe algumas das interfaces disponíveis no projeto “Consulta Opinião”, utilizando o banco de dados NoSQL baseado em documento MongoDB na camada de persistência de dados. A interface móvel apresenta, através de um mapa, os estabelecimentos públicos cadastrados no sistema que podem ser avaliados por usuários voluntários. Ao selecionar um dos estabelecimentos para avaliação, o usuário deverá preencher um questionário sobre a qualidade de atendimento, infraestrutura, e outras questões relevantes para avaliação dos serviços prestados pela instituição pública. A interface *web* apresenta a nota média de cada estabelecimento público cadastrado, fazendo uma classificação por cores das instituições avaliadas, onde verde significa que o estabelecimento público foi considerado bom ou ótimo; e vermelho quando a instituição foi considerada ruim ou regular.

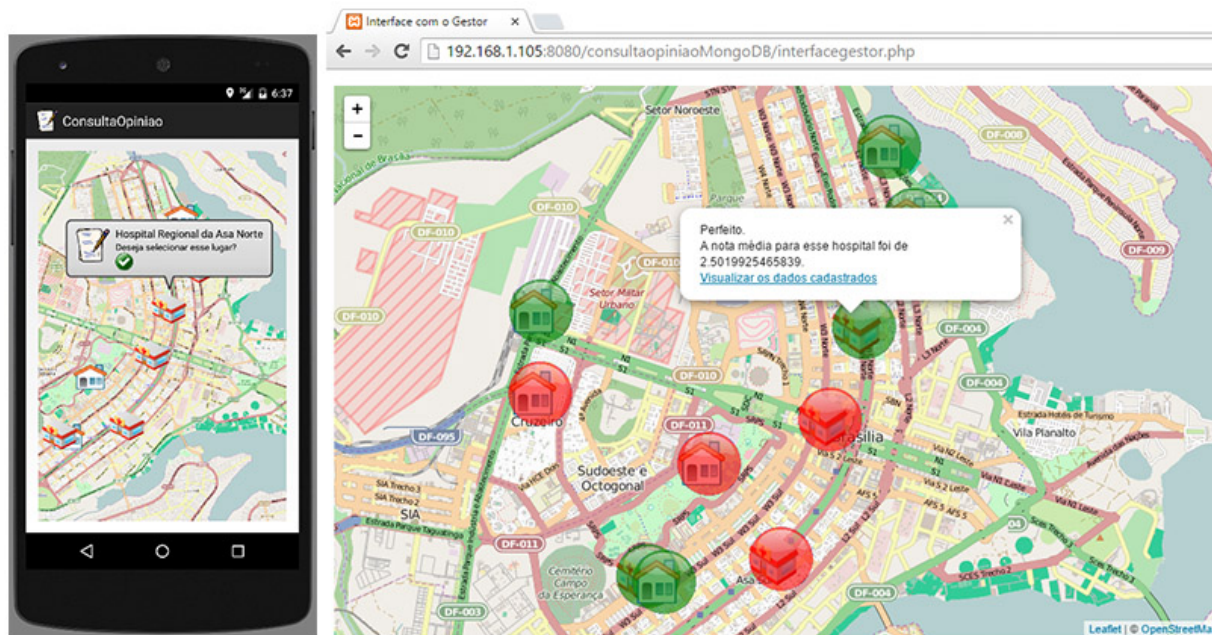


Figura 4.1: Interface móvel e *web* da aplicação Consulta Opinião utilizando o MongoDB na camada de persistência de dados.

Devido a mudança do banco de dados relacional para um banco de dados não relacional, houve, conseqüentemente, a necessidade de refazer a modelagem de dados a fim de proporcionar o armazenamento eficiente das informações do sistema em um banco de dados NoSQL. O modelo de dados em projetos não relacionais utilizam a estratégia da desnormalização, que incorpora dados de diferentes tabelas do modelo relacional em um único documento no modelo não relacional. Na desnormalização, também é possível realizar uma redundância controlada dos dados para que o sistema distribuído em dife-

rentes máquinas (*cluster*) não necessite requisitar informações a outros nós participantes do conjunto. A estratégia da desnormalização é utilizada para evitar que uma consulta busque dados em diferentes máquinas, o que pode gerar tráfego de informações na rede e inconsistência de informações. Assim, cria-se um documento mais complexo, que possui todas as informações necessárias para atender a uma determinada requisição do usuário, mas que não comprometa o desempenho do sistema.

A Figura 4.2 exibe o modelo relacional do banco de dados central do projeto Consulta Opinião em sua arquitetura original. O modelo é composto por cinco tabelas: Usuario, Estabelecimento, TipoEstabelecimento, Questionario e Avaliacao. A tabela Usuario armazena as informações do usuário do aplicativo. A tabela Questionario armazena as perguntas cadastradas para cada tipo de estabelecimento do sistema. As tabelas Estabelecimento e TipoEstabelecimento armazenam informações sobre as instituições públicas cadastradas no sistema, que poderão ser avaliadas por usuários voluntários. Na tabela Estabelecimento é possível encontrar os campos latitude e longitude para armazenar as coordenadas geográficas da instituição pública cadastrada. A tabela Avaliacao armazena as notas das respostas dadas pelos usuários voluntários, para cada pergunta relativa aos estabelecimentos avaliados.

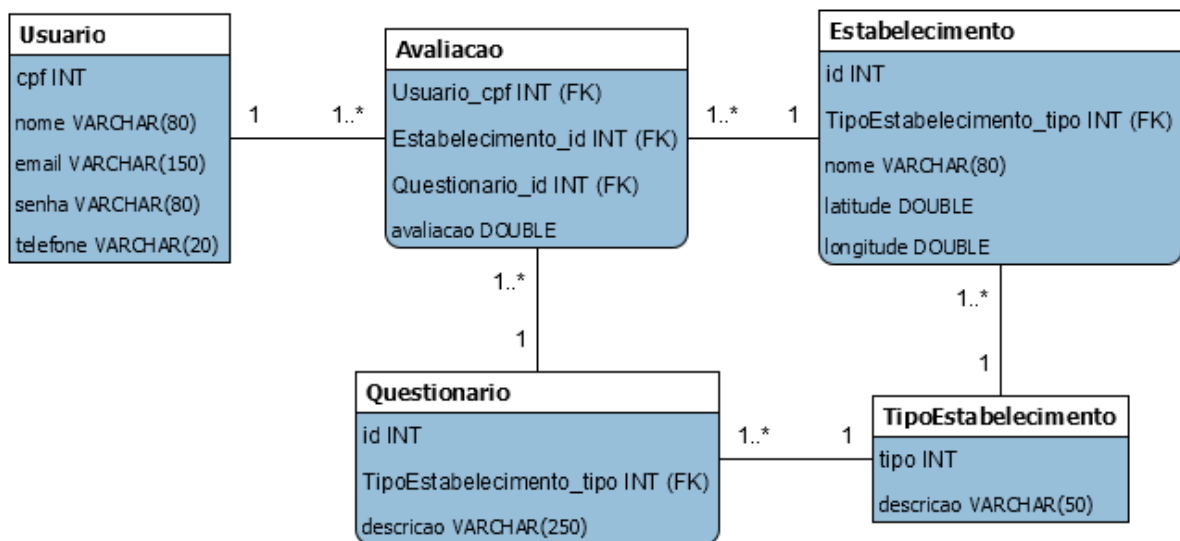


Figura 4.2: Modelo relacional do projeto Consulta Opinião [11].

A Figura 4.3 apresenta duas versões do modelo não relacional para o banco de dados do projeto Consulta Opinião adaptadas para o banco de dados NoSQL baseado em documento MongoDB usando a notação especificada em Vera *et al* [60]. Em ambas versões, o modelo passou a armazenar quatro coleções (comparadas às tabelas no modelo relacional) chamadas Usuário, Estabelecimento, Questionário e Avaliação. Semelhantemente a tabela Usuário, a coleção Usuário armazena as informações do usuário do aplicativo.

A coleção Questionário também tem a mesma função da tabela Questionario (modelo relacional) e armazena as perguntas cadastradas para cada tipo de estabelecimento. A coleção Estabelecimento armazena as informações das instituições públicas que poderão ser avaliadas. Nesta coleção são armazenados dados geográficos, mais especificamente o objeto geométrico ponto referenciando as coordenadas geográficas (longitude e latitude) do estabelecimento público cadastrado. A diferença de uma versão do modelo não relacional para outra, está na forma de armazenar os dados da avaliação feita pelo usuário voluntário. Na primeira versão (Figura 4.3(a)), a coleção Avaliação incorpora dados das coleções Estabelecimento e Usuário, além da nota média de avaliação para cada estabelecimento. Essa estratégia busca fazer uma redundância controlada dos dados, com o intuito de melhorar o desempenho das consultas e facilitar a distribuição dos dados em uma possível implementação em ambiente de *cluster*. Já na segunda versão (Figura 4.3(b)), a coleção Avaliação somente faz referência às coleções Estabelecimento e Usuário, através do código de identificação das referidas coleções, e registra a nota média de avaliação do estabelecimento.

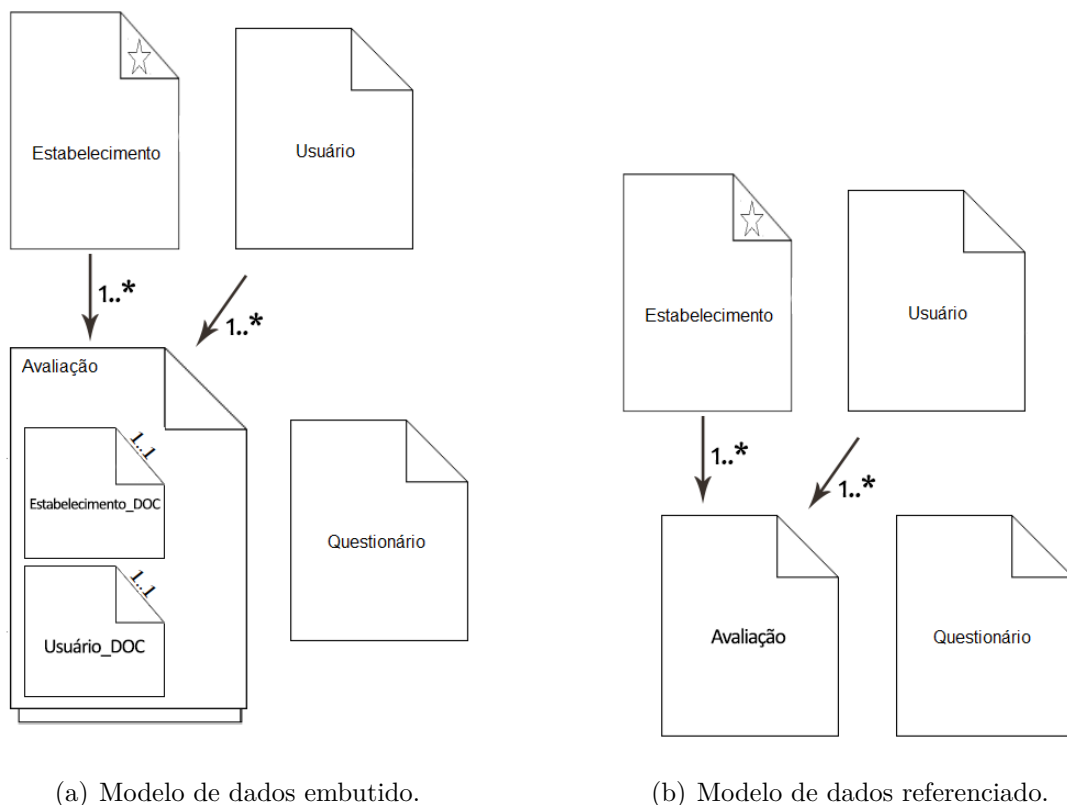


Figura 4.3: Versões do modelo não relacional do projeto Consulta Opinião.

Segundo a notação de Vera *et al* [60] a estrela à direita acima da coleção Estabelecimento diz que os documentos possuem um objeto geográfico do tipo ponto. As setas

ligando Estabelecimento para Avaliação, e Usuário para Avaliação, representam que os documentos Estabelecimento e Usuário serão referenciados no documento Avaliação.

O Código 4.1 apresenta a estrutura do documento *Usuário* responsável por armazenar as informações cadastrais do usuário voluntário, como nome, e-mail, senha, telefone e CPF.

---

```
1 {
2   "_id": <objeto ID>,
3   "cpf": "<numero>",
4   "nome": "<caractere>",
5   "email": "<caractere>",
6   "senha": "<caractere>",
7   "telefone": "<caractere>"
8 }
```

---

Código 4.1: Estrutura do documento JSON: Usuario.

O Código 4.2 apresenta a estrutura do documento *Questionário*, que define suas questões a partir do tipo de estabelecimento a ser avaliado. Por exemplo, o tipo “Escola” tem questões referentes a avaliação de instituições de ensino, devendo ser essas questões listadas no campo *questoes*.

---

```
1 {
2   "_id": <objeto ID>,
3   "tipo": "<caractere>",
4   "questoes": [
5     "<caractere>",
6     "<caractere>",
7     "<caractere>"
8   ]
9 }
```

---

Código 4.2: Estrutura do documento JSON: Questionario.

O Código 4.3 apresenta a estrutura do documento *Estabelecimento*, em que se encontra os dados cadastrais do estabelecimento público a ser avaliado, como código, nome e tipo de estabelecimento, além dos dados de sua localização geográfica estruturados no formato GeoJSON (linhas 6 a 9).

---

```
1 {
2   "_id": <objeto ID>,
3   "codigo": "<caractere>",
4   "nome": "<caractere>",
5   "tipo": "<caractere>",
6   "localizacao": {
7     "type": "<tipo objeto geografico>",
```

```
8     "coordinates": [<longitudo >, <latitude >]
9   }
10 }
```

---

Código 4.3: Estrutura do documento JSON/GeoJSON: Estabelecimento.

O Código 4.4 apresenta a estrutura do documento *Avaliação*, em sua primeira versão, conforme mostrado na Figura 4.3(a), que armazena a nota média atribuída pelo usuário voluntário a um estabelecimento público. Neste documento, dados relativos ao estabelecimento e usuário são incorporados, a fim de melhorar o desempenho do sistema em ambientes distribuídos.

---

```
1 {
2   "_id": <objeto ID>,
3   "estabelecimento": {
4     "codigo": "<caractere>",
5     "nome": "<caractere>",
6     "tipo": "<caractere>"
7   },
8   "usuario": {
9     "cpf": "<numero>",
10    "nome": "<caractere>",
11    "email": "<caractere>"
12  },
13  "avaliacao": <numero>
14 }
```

---

Código 4.4: Estrutura do documento JSON: Avaliacao (Estabelecimento e Usuario embutidos).

O Código 4.5 apresenta a estrutura do documento *Avaliação*, em sua segunda versão, conforme mostrado na Figura 4.3(b), que referencia documentos das coleções Estabelecimento e Usuário, e armazena a nota média atribuída para um estabelecimento público.

---

```
1 {
2   "_id": <objeto ID>,
3   "estabelecimento_id": "<caractere>",
4   "usuario_id": "<numero>",
5   "avaliacao": <numero>
6 }
```

---

Código 4.5: Estrutura do documento JSON: Avaliacao (Estabelecimento e Usuario referenciados).

Para testes iniciais, foram cadastrados alguns usuários, questionários e estabelecimentos (escolas e hospitais). Em seguida, para realizar as avaliações dos estabelecimentos

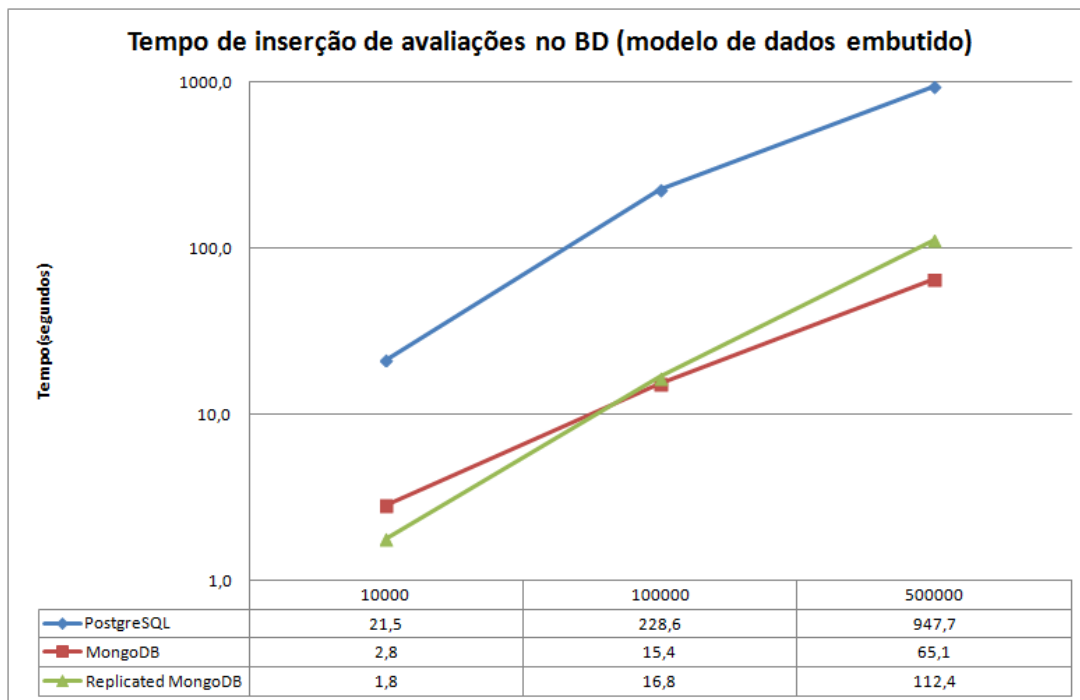
cadastrados foi desenvolvido um simulador de avaliação, que gera uma nota aleatória para cada quesito avaliado do estabelecimento e faz a inserção dos dados no SGBD. O simulador funciona da seguinte forma:

1. Um usuário cadastrado no banco de dados é selecionado aleatoriamente para realizar a avaliação do estabelecimento.
2. Um estabelecimento público cadastrado também é selecionado aleatoriamente para ser avaliado.
3. O simulador gera uma nota aleatória para cada questão a ser avaliada no estabelecimento e, em seguida, calcula uma média final, a partir das notas geradas, para avaliação da instituição.
4. Armazena os dados no banco de dados.

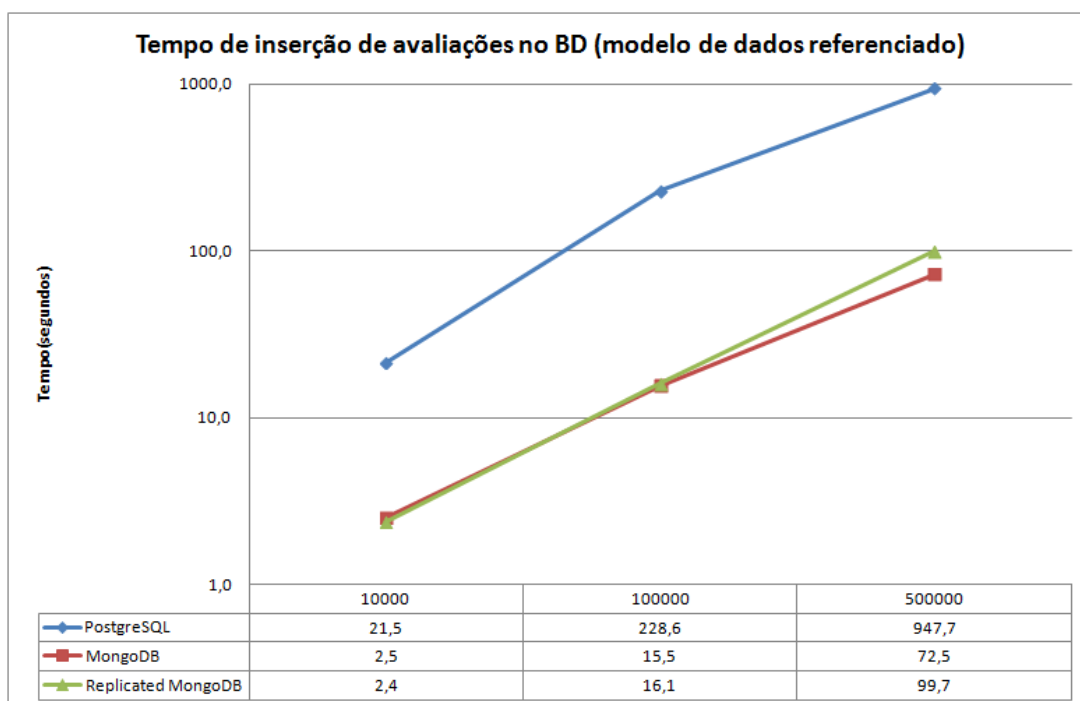
Os *scripts* de teste foram desenvolvidos na linguagem PHP, e possuem um mecanismo de registro do tempo da execução do *script* para avaliar o desempenho do banco de dados relacional PostgreSQL 9.3 e do banco de dados NoSQL baseado em documento MongoDB 3.0.3. A arquitetura de armazenamento de dados do Consulta Opinião foi implementada de duas formas: utilizando um único servidor (PostgreSQL e MongoDB) e utilizando um conjunto de servidores replicados (MongoDB). Os testes executados em ambiente de processamento local – servidor único – utilizaram um computador com as seguintes especificações: processador Intel Core i5 2.5GHz, 6GB de memória RAM, disco SATA de 1TB de armazenamento, executando Sistema Operacional Windows 8 64-bit. Os testes executados em ambiente de *cluster* – servidores replicados – utilizaram 3 máquinas com as seguintes especificações: processador AMD Phenom II X2 3.2GHz, 4GB de memória RAM, disco SATA de 500GB de armazenamento, executando o Sistema Operacional Windows 7 Professional 64-bit, rede ethernet 10/100 Mbps.

Dois tipos de testes foram implementados: inserção e leitura. O teste de inserção é para simular a inserção de um grande número de avaliações de estabelecimentos públicos através da geração de dados aleatórios por computador. O teste de leitura lista os estabelecimentos públicos cadastrados no banco de dados, gerando uma nota média para cada instituição usando uma consulta de agregação das avaliações, que pode ser considerada complexa.

A Figura 4.4 mostra a comparação dos tempos registrados na inserção de dados referentes às avaliações dos estabelecimentos públicos. A Figura 4.4(a) apresenta o resultado das inserções das avaliações utilizando o modelo de dados embutido; e a Figura 4.4(b) apresenta o resultado das inserções das avaliações utilizando o modelo de dados referenciado.



(a) MongoDB utilizando modelo de dados embutido.



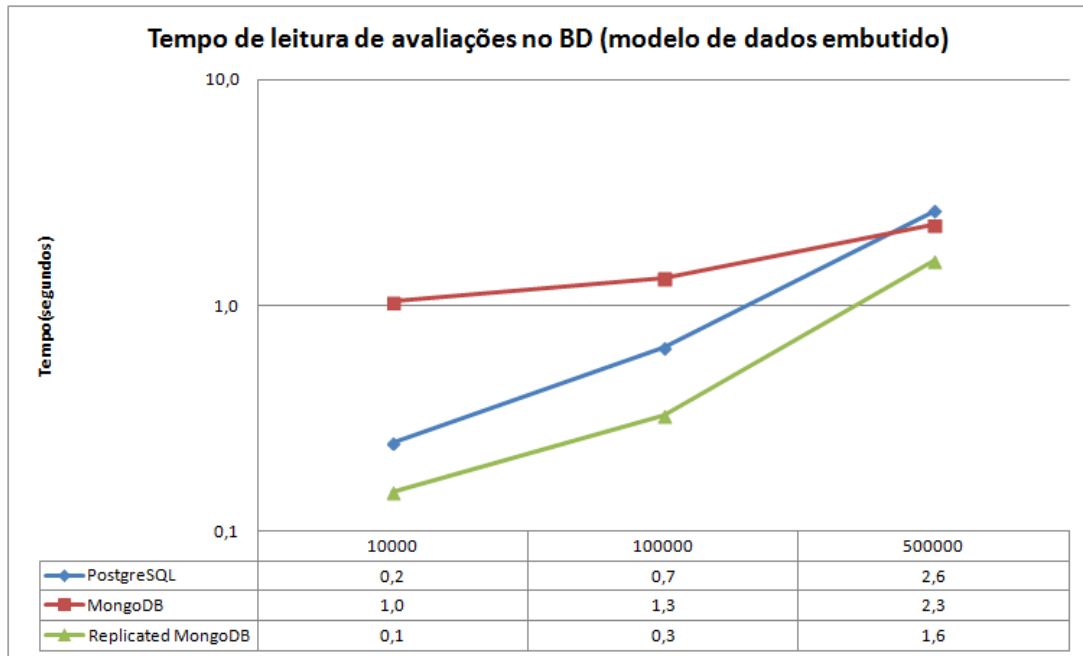
(b) MongoDB utilizando modelo de dados referenciado.

Figura 4.4: Comparação dos tempos de inserção dos dados de avaliação nos bancos de dados PostgreSQL e MongoDB.

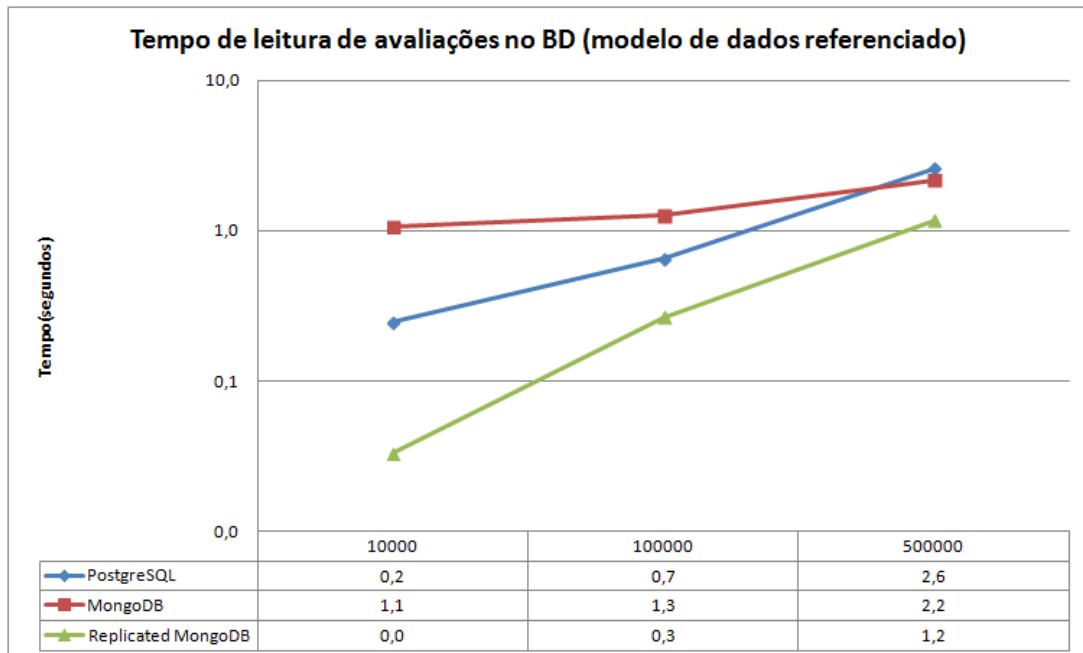
A Figura 4.5 apresenta a comparação dos tempos aferidos nos testes de leitura dos dados das avaliações armazenadas nos bancos de dados PostgreSQL e MongoDB. A Fi-



Figura 4.5(a) apresenta os tempos necessários para realizar a leitura das avaliações utilizando o modelo de dados embutido; e a Figura 4.5(b) os tempos necessários para realizar a leitura das avaliações utilizando o modelo de dados referenciado.



(a) MongoDB utilizando modelo de dados embutido.



(b) MongoDB utilizando modelo de dados referenciado.

Figura 4.5: Comparação dos tempos de leitura dos dados de avaliação nos bancos de dados PostgreSQL e MongoDB.

Cada teste possui suas particularidades, por exemplo, o teste de inserção gera dados

aleatórios para que seja possível fazer uma avaliação do desempenho dos bancos de dados PostgreSQL e MongoDB na escrita de dados. Para isso, realizou-se baterias de simulações que armazenaram, respectivamente, 10.000, 100.000 e 500.000 documentos ou dados de avaliação de estabelecimentos públicos. A Figura 4.4 deixa claro que o MongoDB apresenta desempenho superior ao PostgreSQL em todos os testes de inserção de dados realizados. Analisando os tempos registrados na inserção de dados, o MongoDB obteve um desempenho 14 vezes superior ao PostgreSQL quando o número de documentos ultrapassou a marca de 100.000 inserções. A diferença nos tempos registrados para inserção de dados de avaliações, para os modelos de dados embutido (Figura 4.4(a)) e referenciado (Figura 4.4(b)), foi mínima, demonstrando uma certa semelhança entre os modelos.

O armazenamento de dados no MongoDB é realizado através de uma *engine*, responsável pela gestão dos dados do banco, incluindo o gerenciamento da memória. É possível configurar uma variedade de *engines* de armazenamento, permitindo escolher a mais adequada de acordo com a aplicação em questão. Neste caso, utilizou-se a *engine* “*WiredTiger*”, que fornece um modelo de concorrência a nível de documento, pontos de verificação, compressão, entre outros recursos. Esta *engine* é recomendável para a maioria dos casos, pois geralmente apresenta melhores desempenhos e possui índices compactos<sup>1</sup>.

Por outro lado, a Figura 4.5 mostra um melhor desempenho do banco de dados MongoDB (replicado), em ambos modelos de dados utilizados. Neste caso, também há uma mínima diferença entre os tempos registrados nos testes de leitura dos modelos de dados embutido (Figura 4.5(a)) e referenciado (Figura 4.5(b)). O PostgreSQL apresenta melhor desempenho em operações de leitura, em relação ao MongoDB com único servidor, até que o número de avaliações armazenadas exceda aproximadamente 400.000 documentos. A partir desse número de documentos armazenados, o MongoDB, com único servidor, também apresenta melhor desempenho em comparação com o PostgreSQL. Para os testes de leitura, foram realizadas operações de leitura e agregação dos dados referentes às notas geradas para os estabelecimentos avaliados. Os testes de leitura foram executados quando o banco de dados possuía, respectivamente, 10.000, 100.000 e 500.000 avaliações cadastradas.

Pelos resultados apresentados, o MongoDB se destaca principalmente pela agilidade no armazenamento de dados, foco deste trabalho de dissertação, e quando a quantidade de informações armazenadas é consideravelmente grande para obter melhor desempenho nas leituras de dados em comparação ao banco de dados PostgreSQL. Os resultados demonstraram também que a replicação dos dados em um *cluster* foi uma boa alternativa, tanto para inserção quanto para leitura de dados.

---

<sup>1</sup><https://docs.mongodb.com/manual/>

## 4.2 Prova de Conceito 2: Comune

O Comune é um aplicativo para a plataforma Android que possibilita a aplicação de questionários aos seus usuários bem como a coleta de relatos feitos por esses. A proposta é que o poder público – representado por Agências, Secretarias e demais instituições governamentais – seja o aplicador dos questionários, além de ser o responsável por definir quais pesquisas e perguntas serão feitas aos usuários de seus serviços. Já os usuários que responderão às pesquisas devem ser cidadãos que utilizam periodicamente algum serviço público cadastrado no sistema. A cada pesquisa respondida, é gerado um conjunto de dados que poderá auxiliar no processo de planejamento estratégico feito pelos administradores do serviço.

O Comune [19] é a evolução do aplicativo Consulta Opinião [11], apresentado na primeira prova de conceito. As tarefas definidas para o Comune abrangeram a reformulação da interface do Consulta Opinião, a possibilidade de trabalhar *off-line* e com dados de imagem e vídeo, além da implementação de um novo banco de dados relacional e a disponibilização de *web services* para acesso e manipulação dos dados. Tal projeto consiste na criação de um amplo sistema para coleta de dados e de *feedbacks* que colaborem na elaboração de políticas públicas mais eficazes e eficientes, contando com:

- Aplicativos móveis para o sistema operacional Android, que possibilitarão a aplicação de questionários e coleta de sugestões ou reclamações.
- Um banco de dados central para todos os sistemas cliente.

A Figura 4.6 apresenta algumas telas do Comune, referente a exibição dos estabelecimentos cadastrados, busca de serviços públicos para avaliação e lista de relatos do usuário. A tela que faz a exibição dos estabelecimentos cadastrados mostra ícones identificando o tipo do estabelecimento assim como mostrando sua localização geográfica no mapa. Ao selecionar um dos estabelecimentos exibidos, o usuário pode ver informações úteis sobre aquele órgão público, como o horário de funcionamento. O aplicativo permite que o usuário pesquise por estabelecimentos localizados próximos a um determinado raio de distância das coordenadas geográficas do aparelho celular do voluntário. Além disso, é possível realizar relatos, incluindo arquivos de imagem ou vídeo, e avaliações dos estabelecimentos cadastrados.

Como na primeira prova de conceito, o Comune foi projetado para utilizar o SGBD relacional PostgreSQL, e posteriormente foi adaptado para comunicar-se com os bancos de dados NoSQL baseados em documento MongoDB e CouchDB. A comunicação entre o banco de dados, presente na camada de persistência de dados, e o aplicativo ocorre através de *web services*, que precisaram ser reprogramados para suportar os novos tipos de banco de dados.

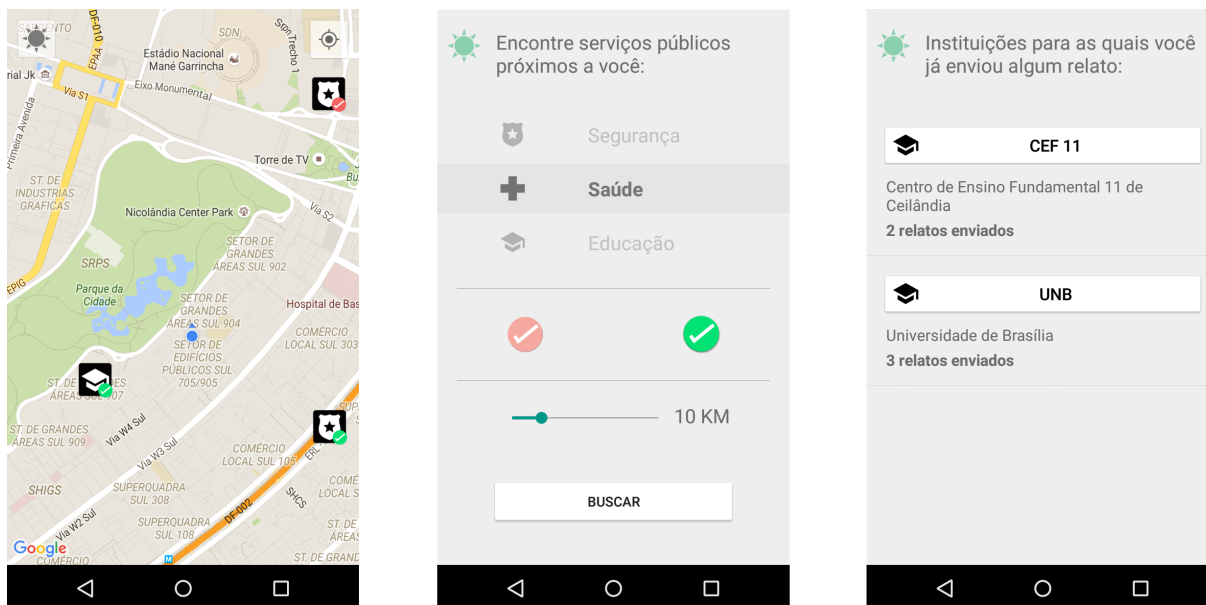


Figura 4.6: Telas do aplicativo Comune [19].

Como um dos requisitos do Comune é ter a capacidade de funcionar *off-line*, o aplicativo possui um banco de dados local que é armazenado no aparelho celular do usuário do serviço público. Este banco local utiliza a biblioteca SQLite do Android, e possui um total de 14 tabelas. Localmente são armazenados dados de usuários conectados, pesquisas e respostas dadas pelo usuário, relatos feitos e informações sobre estabelecimentos. Já o banco de dados central, localizado remotamente para atendimento de todos os usuários do aplicativo, armazena as mesmas informações, porém com detalhes adicionais. O modelo relacional do banco de dados central do Comune possui 24 tabelas. A Figura 4.7 apresenta um modelo relacional resumido do banco de dados central do Comune, apresentando 9 tabelas, sendo elas responsáveis por realizar o armazenamento dos dados dos estabelecimentos públicos, pesquisas e das questões das pesquisas. O modelo relacional completo é apresentado em De Branco [19].

Similarmente à prova de conceito anterior, houve a necessidade de remodelar o banco de dados central para adaptá-lo ao banco de dados NoSQL baseado em documento. Para esta remodelagem, utilizou-se novamente a notação de Vera *et al* [60], conforme apresentado na Figura 4.8. O modelo não relacional apresentado utiliza o conceito de orientação agregada para formação de suas estruturas de dados. A orientação agregada manipula dados em unidades que possuem estruturas mais complexas que um conjunto de tuplas (registro do banco de dados). Dessa forma, o número de coleções pode ser menor devido a estrutura mais complexa e abrangente do agregado. Neste caso, a estrutura em agregado contribui para manipulação de dados e gerenciamento da consistência, sendo mais fácil de se trabalhar com banco de dados em *cluster*, já que os agregados são unidades naturais

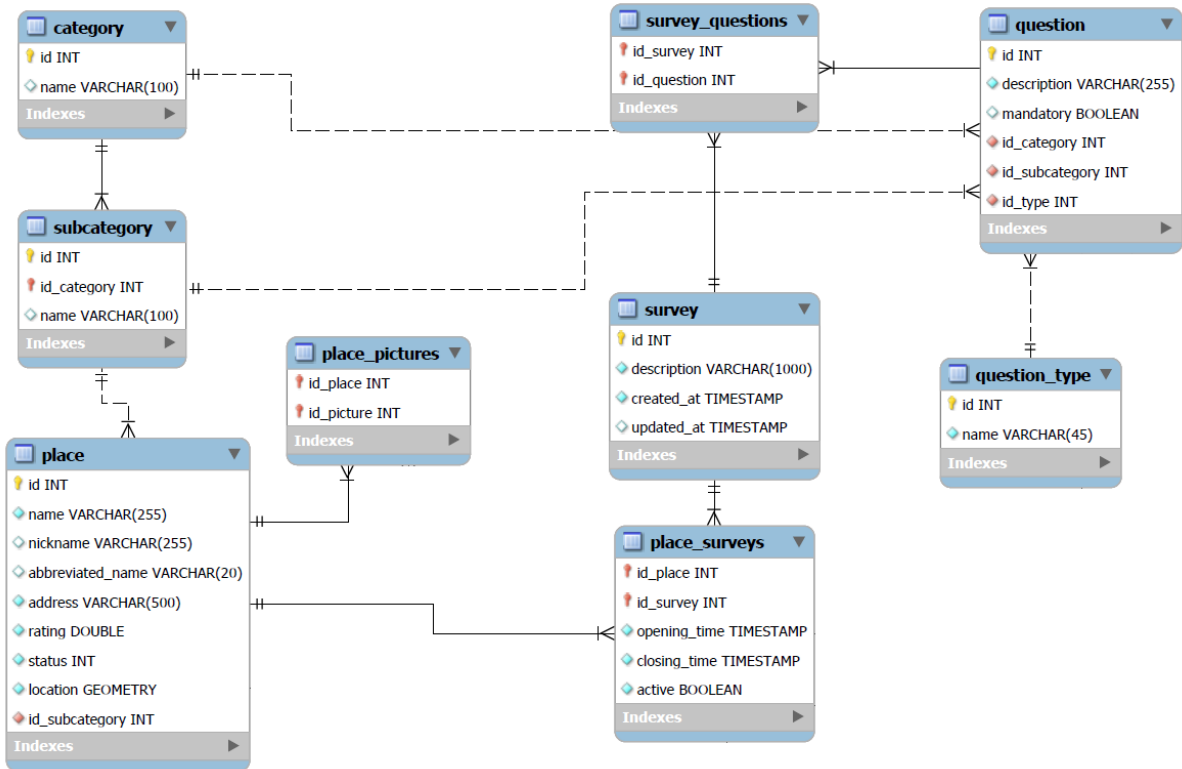


Figura 4.7: Modelo relacional resumido do banco de dados central Comune.

para replicação e fragmentação [52].

Neste modelo não relacional tem-se a representação de 6 coleções de documentos e 4 documentos embutidos. Neste caso, têm-se as seguintes coleções: *User*, *Place*, *Report*, *Survey*, *SurveyAnswer* e *Fs.files*. As estruturas não listadas fazem parte da composição agregada das principais coleções. A coleção *User* armazena as informações dos usuários voluntários do aplicativo. A coleção *Place* armazena os dados referentes aos estabelecimentos públicos (locais) cadastrados no sistema e define quais serão as pesquisas disponíveis aos usuários do estabelecimento. A coleção *Report* armazena os dados dos relatos feitos pelos usuários para os estabelecimentos públicos e seus respectivos comentários. A coleção *Survey* armazena as pesquisas - incluindo questões - que podem ser disponibilizadas aos estabelecimentos públicos. A coleção *SurveyAnswer* armazena os dados de resposta, para cada questão, das pesquisas salvas por usuários voluntários. Finalmente, a coleção *Fs.Files* grava os metadados dos arquivos de imagem, áudio e vídeo armazenados no banco de dados.

O Código 4.6 apresenta a estrutura do documento *Place*, que armazena informações cadastrais dos estabelecimentos públicos. Neste documento é possível encontrar um objeto GeoJSON embutido que representa uma geometria utilizando as coordenadas geográficas de localização do estabelecimento (linhas 9 a 12). O Código 4.7 apresenta o documento

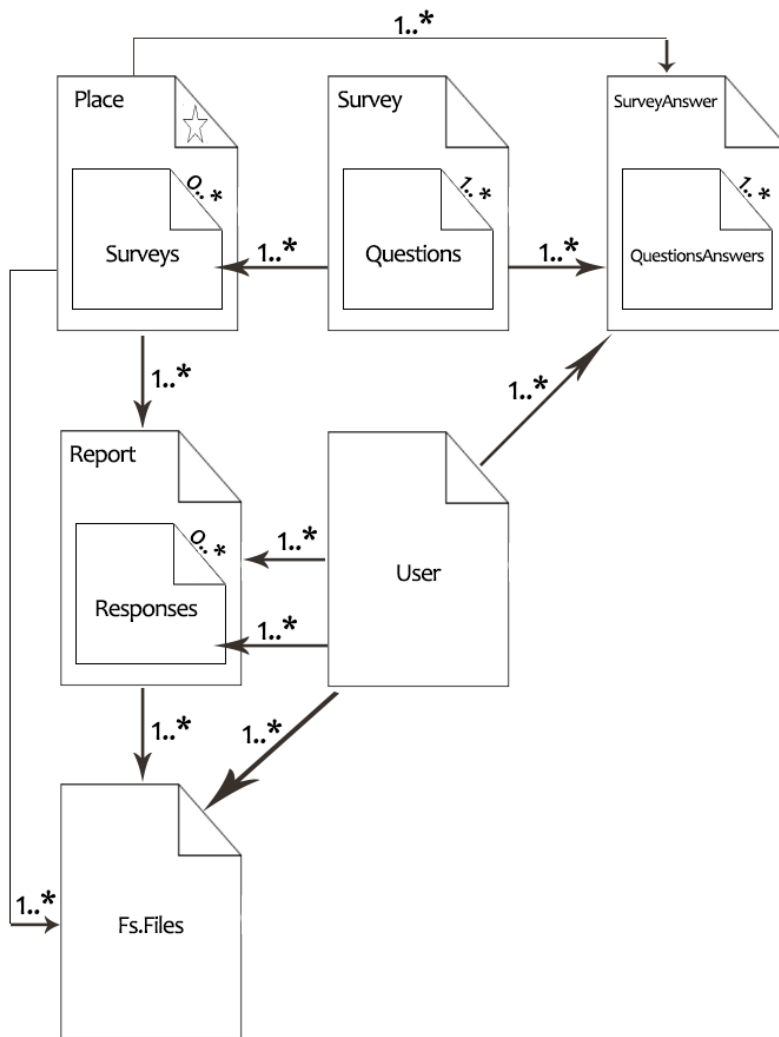


Figura 4.8: Modelo não relacional banco de dados Comune.

*Fs.Files* onde fica gravado os metadados dos arquivos binários armazenados no sistema de banco de dados. A estrutura dos documentos *User*, *Report*, *Survey* e *SurveyAnswer* é apresentada no Apêndice A desta dissertação.

---

```

1 {
2   "id": <objeto ID>,
3   "name": "<caractere>",
4   "nickname": "<caractere>",
5   "abbrevName": "<caractere>",
6   "address": "<caractere>",
7   "rating": <numero>,
8   "status": <numero>,
9   "location": {
10     "type": "<tipo objeto geografico>",
11     "coordinates": [<longitude>, <latitude>]
  
```

```

12     },
13     "category": <numero>,
14     "descCategory": "<caractere>",
15     "subcategory": <numero>,
16     "descSubCategory": "<caractere>",
17     "surveys": [{
18         "idSurvey": <numero>,
19         "openingTime": "<data-hora>",
20         "closingTime": "<data-hora>",
21         "active": <boolean>
22     }],
23     "workingDays": [{
24         "dayOfTheWeek": <numero>,
25         "openingTime": "<hora>",
26         "closingTime": "<hora>"
27     }, {
28         "dayOfTheWeek": <numero>,
29         "openingTime": "<hora>",
30         "closingTime": "<hora>"
31     }]
32 }

```

---

Código 4.6: Estrutura do documento Place do BD Comune.

---

```

1 {
2     "idReport": <numero>,
3     "idPlace": <numero>,
4     "idUser": <numero>,
5     "Content-Type": "<caractere>",
6     "filename": "<caractere>",
7     "extension": "<caractere>",
8     "uploadDate": "<data-hora>",
9     "length": <numero>,
10    "chunkSize": <numero>,
11    "md5": "<caractere>"
12 }

```

---

Código 4.7: Estrutura do documento Fs.Files do BD Comune.

---

Para validação da arquitetura foram implementados 32 *web services*, organizados em 4 classes: *users*, *places*, *surveys* e *reports*. A seguir são apresentados os principais *web services* implementados para o Comune, que são: *places/registerNewPlace*; *places/getNearbyPlaces*; *places/findNearbyPlaces*; *reports/uploadReportPicture* e *reports/uploadReportAudio*. No Apêndice B pode-se encontrar todos os 32 *web services* implementados e suas respectivas descrições.

## Principais *web services* implementados para a arquitetura do projeto **Comune**

- `places/registerNewPlace`: Serviço para cadastro de estabelecimento público.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Nome, apelido, nome abreviado, endereço, longitude, latitude, categoria do estabelecimento.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com código de identificação, longitude e latitude do estabelecimento público criado. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
  - Observações: Este serviço armazena dados convencionais e utiliza um objeto geométrico ponto, adotando a notação GeoJSON e as coordenadas de longitude e latitude, para armazenar a localização do estabelecimento público.
- `places/getNearbyPlaces`: Serviço para retornar uma lista de estabelecimentos públicos próximos a determinado raio de distância.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: GET
  - Entrada: Longitude, Latitude, e raio de distância a ser pesquisado.
  - Retorno: Caso exista estabelecimentos públicos próximos, retorna um objeto JSON com uma lista dos estabelecimentos. Caso não exista estabelecimentos públicos próximos, retorna um objeto JSON vazio.
  - Observações: Este serviço realiza uma consulta espacial no banco de dados para retornar os estabelecimentos públicos próximos às coordenadas geográficas do aparelho celular do usuário voluntário.
- `places/findNearbyPlaces`: Serviço para retornar uma lista de estabelecimentos públicos próximos a determinado raio de distância e que pertençam a determinada categoria.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: GET
  - Entrada: Longitude, Latitude, raio de distância e categoria.



- Retorno: Caso exista estabelecimentos públicos que atendam os critérios estabelecidos, retorna um objeto JSON com uma lista dos estabelecimentos. Caso não exista estabelecimentos públicos que atendam os critérios estabelecidos, retorna um objeto JSON vazio.
- Observações: Este serviço realiza uma consulta espacial no banco de dados para retornar os estabelecimentos públicos próximos às coordenadas geográficas do aparelho celular do usuário voluntário.
- reports/uploadReportPicture: Serviço para armazenar uma imagem do relato para o estabelecimento público.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação do relato e arquivo de imagem.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com código de identificação da imagem armazenada. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
- reports/uploadReportAudio: Serviço para armazenar um áudio do relato para o estabelecimento público.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação do relato e arquivo de áudio.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com código de identificação do áudio armazenado. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.

A implementação da arquitetura de armazenamento de dados do Comune foi realizada de três formas diferentes: modo *standalone* (único servidor), replicada e fragmentada. Inicialmente, implementou-se a arquitetura utilizando um único servidor de banco de dados adotando as tecnologias PostgreSQL, MongoDB e CouchDB na camada de persistência de dados.

Após a configuração do ambiente com um único servidor, foram realizados testes para comparação dos tempos de inserção de dados convencionais, dados multimídia e dados geográficos utilizando os bancos de dados PostgreSQL, MongoDB e CouchDB. Os resultados dos testes são apresentados e analisados a seguir para comparação com os outros ambientes computacionais implementados.

Em seguida, implementou-se a arquitetura utilizando um servidor de banco de dados MongoDB replicado, configurado com 3 máquinas, conforme apresentado na Figura 3.5. Essa estratégia foi utilizada para prover redundância e aumentar a disponibilidade dos dados. Nesta implementação, o MongoDB foi configurado para automaticamente eleger um novo nó mestre em caso de falha do nó principal. Isso acontece quando um nó mestre não se comunica com os outros nós por mais de 10 segundos. Com isso, a disponibilidade da aplicação não fica comprometida, mesmo havendo uma falha em um dos servidores de banco de dados.

Algumas configurações específicas também colaboram para que o banco de dados possa se recuperar de falhas ou de encerramentos abruptos, como é o caso da habilitação do registro do *log journal* que serve para requerer o reconhecimento das operações de escrita e certificar que os dados foram gravados em disco no banco de dados<sup>2</sup>. A obrigatoriedade de reconhecer as operações de escrita torna a arquitetura mais confiável. Outra configuração importante para o armazenamento de dados é a definição de uma propriedade chamada *write concern*, que define o nível de reconhecimento das operações de escrita para um único servidor ou conjunto de réplicas (*replica sets*) ou *sharded clusters* (*clusters* fragmentados)<sup>3</sup>. Com objetivo que obter maior confiabilidade no armazenamento de dados utilizou-se, para esta configuração, o valor *majority* que define que a maioria dos servidores participantes do *cluster* devem reconhecer a operação de escrita do usuário da aplicação.

Após os testes da arquitetura de armazenamento utilizando o banco de dados MongoDB replicado, fez-se a implementação da arquitetura utilizando um *cluster* MongoDB fragmentado com 10 máquinas, sendo: 1 máquina (*router*); 3 máquinas para servidores de configuração; e 6 máquinas para servidores de banco de dados, conforme apresentado na Figura 3.6. Detalhes da implementação do ambiente computacional dos bancos de dados replicados e fragmentados foram citados na Seção 3.4 desta dissertação.

#### 4.2.1 Análise dos Testes de Inserção de Dados das Implementações da Arquitetura

Para realizar testes no Comune, foram desenvolvidos novos *scripts* em PHP para simular o armazenamento de uma grande quantidade de dados, de diversos formatos, como dados alfanuméricos (convencionais), dados multimídia (imagem, áudio e vídeo) e dados geográficos. A ideia é utilizar os *scripts* PHP para gerar dados aleatórios, que serão armazenados no banco de dados, assim como para obter os tempos de execução das operações de inserção de dados. Cada teste realizado possui 4 execuções, que geram uma sequência de 1.000, 10.000, 100.000 e 500.000 documentos e inserções de banco de dados. A cada

---

<sup>2</sup><https://docs.mongodb.com/v3.0/core/journaling/>

<sup>3</sup><https://docs.mongodb.com/v3.0/reference/write-concern/>

execução, é registrado o tempo de duração, em segundos, das operações de inserção de dados para posterior comparação.

Os testes apresentados a seguir, executados em ambiente de processamento local, utilizaram um computador com as seguintes especificações: processador Intel Core i5 2.5GHz, 6GB de memória RAM, disco SATA de 1TB de armazenamento, executando o Sistema Operacional Windows 10 64-bit. Já os testes executados em ambiente de *cluster* (replicação e fragmentação), utilizaram máquinas com as seguintes especificações: processador AMD Phenom II X2 3.2GHz, 4GB de memória RAM, disco SATA de 500GB de armazenamento, executando o Sistema Operacional Windows 7 Profissional 64-bit, rede ethernet 10/100 Mbps. As versões dos bancos de dados foram as seguintes: PostgreSQL 9.3, MongoDB 3.0.12 e CouchDB 1.6.1.

O teste de armazenamento de dados convencionais foi desenvolvido utilizando dados aleatórios para simular a avaliação de um estabelecimento público por um usuário cadastrado no Comune. Gerou-se uma sequência de dados e respostas aleatórias, mas que condiziam com questões e alternativas disponíveis no questionário. Neste teste somente dados alfanuméricos foram armazenados no banco de dados central do Comune. A Figura 4.9 apresenta os tempos de inserção de dados convencionais no BD Comune, comparando o desempenho dos bancos de dados PostgreSQL, MongoDB e CouchDB, utilizando a arquitetura de um único servidor; e do banco de dados MongoDB replicado e fragmentado, utilizando a arquitetura em *cluster*.

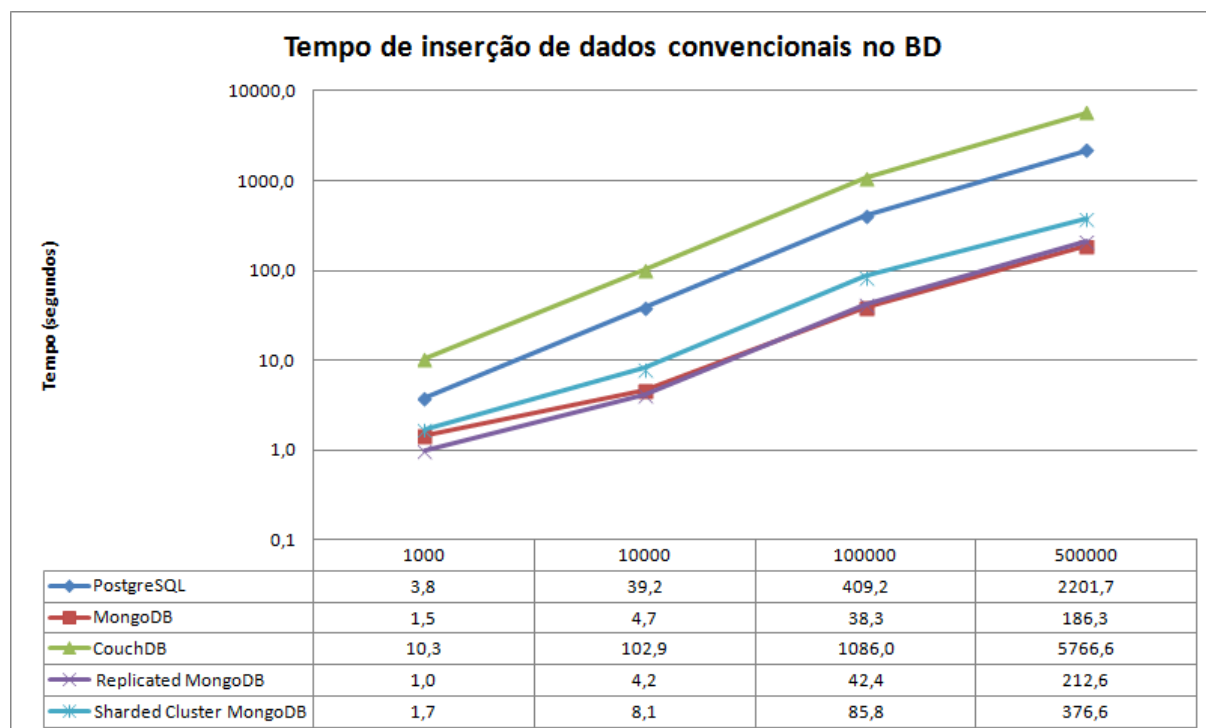


Figura 4.9: Comparativo dos tempos de inserção de dados convencionais no BD Comune.

Os testes de armazenamento de dados convencionais apresentam algumas curiosidades, por exemplo, o tempo de inserção no banco de dados MongoDB replicado é menor até 10.000 inserções de documentos. A partir de 100.000 inserções de documentos o MongoDB utilizando um único servidor apresenta desempenho superior. Acredita-se que a replicação de muitos dados pode reduzir o desempenho das operações de inserção no banco de dados replicado. Ainda, percebe-se que a arquitetura fragmentada (*sharded cluster*) pode apresentar uma melhor confiabilidade e disponibilidade, mas que compromete o desempenho do banco de dados na inserção de dados. A melhor média do tempo de inserção é do servidor de banco de dados MongoDB em sua configuração *standalone*, ou seja, com um único servidor. A média dos tempos registrados na inserção de dados convencionais são os seguintes (medidos em segundos): PostgreSQL (663,5), MongoDB (57,7), CouchDB (1741,5), *replicated* MongoDB (65,0) e *sharded cluster* MongoDB (118,1).

Já o teste de armazenamento de dados multimídia foi realizado utilizando um arquivo de imagem que foi carregado via *upload*, simulando a forma de envio de arquivos do aplicativo do celular para o *web service*, e posteriormente para o banco de dados. O arquivo utilizado para todos os testes de inserção (multimídia) tinha o peso de 76KB. O teste simulou o envio de relatos com foto para um estabelecimento público cadastrado no Comune. No armazenamento de dados multimídia, o banco de dados MongoDB (*standalone*) obteve um desempenho superior aos outros bancos de dados em todos os testes realizados. Neste teste é interessante observar que o banco de dados MongoDB replicado levou um tempo inesperado para o armazenamento de 500.000 documentos. Provavelmente, este problema tenha ocorrido por falhas na rede ou de comunicação com outras máquinas do *cluster*. A Figura 4.10 apresenta os tempos de inserção de dados multimídia no BD Comune comparando o desempenho dos bancos dados dos avaliados. A média dos tempos registrados na inserção de dados multimídia são os seguintes (medidos em segundos): PostgreSQL (2765,3), MongoDB (481,2), CouchDB (2230,4), *replicated* MongoDB (6123,5) e *sharded cluster* MongoDB (1883,7).

O teste de armazenamento de dados geográficos teve como base uma lista de cidades armazenadas em formato GeoJSON, para os bancos de dados MongoDB e CouchDB; e uma lista de cidades armazenadas em uma tabela, para o banco de dados PostgreSQL. Foram cadastrados novos estabelecimentos públicos sendo a localização geográfica do estabelecimento determinada de forma aleatória. Através da Figura 4.11 é possível comparar os tempos de inserção de dados geográficos nos bancos de dados avaliados. Neste teste viu-se um bom desempenho dos bancos de dados MongoDB e PostgreSQL, porém, o MongoDB (*standalone*) apresentou melhor desempenho médio, conforme os tempos registrados (medidos em segundos): PostgreSQL (79,2), MongoDB (28,2), CouchDB (597,5), *replicated* MongoDB (28,6) e *sharded cluster* MongoDB (61,1).

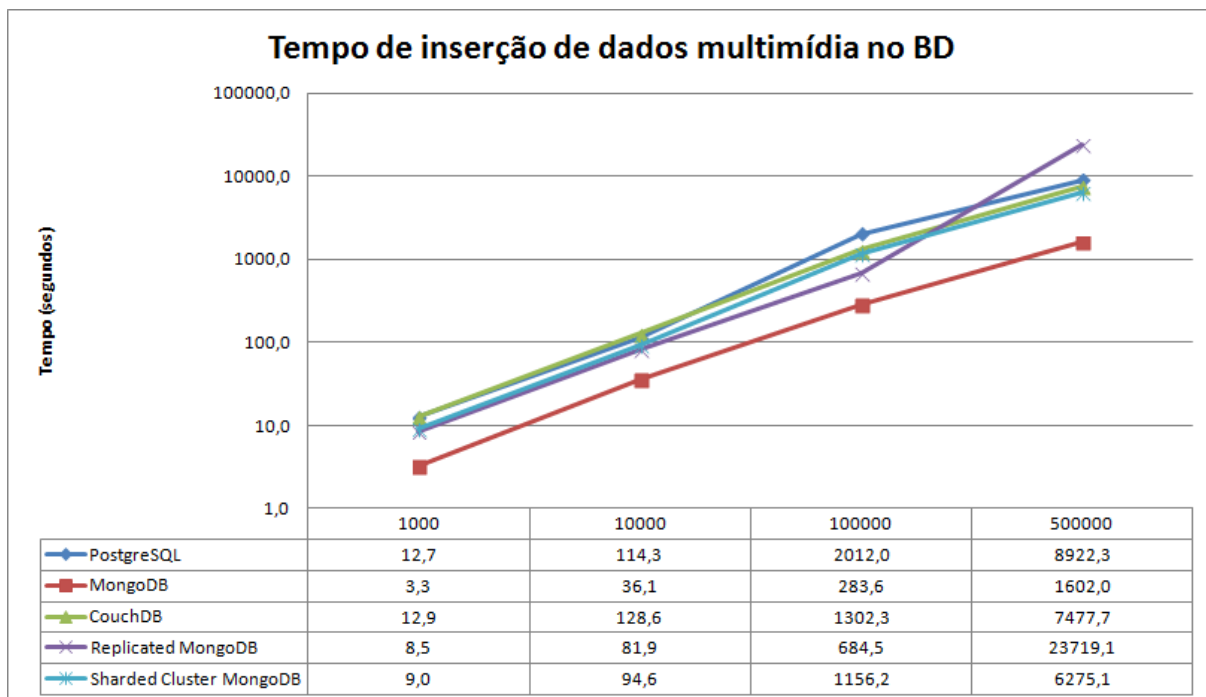


Figura 4.10: Comparativo dos tempos de inserção de dados multimídia no BD Comune.

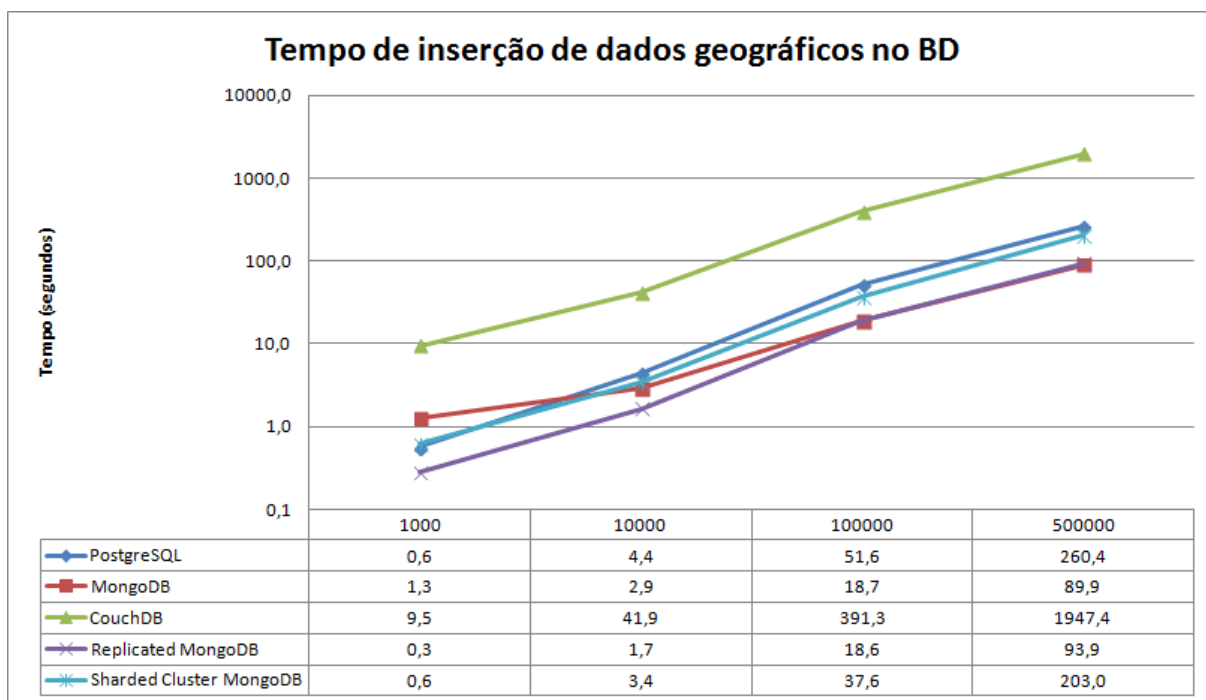


Figura 4.11: Comparativo dos tempos de inserção de dados geográficos no BD Comune.

## 4.2.2 Análise dos Testes de Tolerância a Falhas das Implementações da Arquitetura

Os testes de tolerância a falhas foram realizados fazendo a interrupção de alguns servidores de banco de dados, nas implementações da arquitetura em *cluster* replicada e fragmentada (ver Seção 3.4), utilizando o MongoDB. Nestes testes, não foi-se registrado os tempos de inserção dos dados, mas notou-se o comportamento da execução do servidor de banco de dados, principalmente em relação a sua disponibilidade.

Na implementação replicada da arquitetura, testou-se, inicialmente, a interrupção de um servidor secundário (*slave*), que é responsável pela cópia dos dados do servidor principal (*master*). Neste caso, o servidor de banco de dados principal e o outro servidor secundário continuaram em funcionamento, e a disponibilidade do banco de dados não foi comprometida. Em seguida, ao retirar o outro servidor secundário ainda em funcionamento, o servidor de banco de dados apresentou-se indisponível; mesmo o servidor de banco de dados principal, responsável pela gravação dos dados, estando em funcionamento.

Em outro teste de tolerância a falhas, da implementação replicada da arquitetura, o servidor principal (*master*) foi desligado primeiro, causando uma indisponibilidade momentânea do servidor de banco de dados pelo período aproximado de 4 segundos, até que um servidor secundário fosse eleito o novo servidor principal. Desta forma, as aplicações clientes do servidor de banco de dados, que faziam escritas contínuas na base de dados, foram interrompidas por um curto período de tempo, porém, puderam retomar a gravação dos dados após a recuperação do servidor principal.

Como o conjunto de réplicas, na implementação replicada, utiliza 3 servidores de banco de dados, a tolerância a falhas ficou restrita a interrupção do funcionamento de apenas uma máquina. Assim, a partir do desligamento de uma segunda máquina do conjunto de réplicas, da arquitetura replicada, o servidor de banco de dados fica indisponível.

Na implementação fragmentada da arquitetura, tinha-se a disposição 2 conjuntos de réplicas, com 3 máquinas cada. A fragmentação foi realizada sobre todas as coleções do banco de dados do Comune, no intuito de alcançar alta disponibilidade. Assim, a gravação dos dados puderam acontecer em qualquer um dos fragmentos disponíveis. Para realizar os testes de tolerância a falhas na implementação fragmentada da arquitetura, seguiu-se uma estratégia semelhante a adotada nos testes de tolerância a falhas da implementação replicada. Desse modo, desligou-se, inicialmente, um servidor secundário de um dos fragmentos do *cluster*, onde pode-se notar o normal funcionamento do servidor de banco de dados. Em seguida, retirou-se o outro servidor secundário do mesmo fragmento editado anteriormente. Neste caso, esse fragmento específico ficou impossibilitado de receber operações de escrita, porém, isso não comprometeu a disponibilidade do servidor de banco de

dados, que continuou realizando a gravação de dados a partir do fragmento disponível. No fragmento, até então, não alterado, removeu-se mais um servidor secundário do banco de dados, que continuou em seu normal funcionamento. Somente após a retirada do segundo servidor secundário, do segundo fragmento, é que o banco de dados ficou indisponível.

Em outro teste de tolerância a falhas, da implementação fragmentada da arquitetura, retirou-se, inicialmente, o servidor principal (*master*) de um dos fragmentos do *cluster*. Neste caso, as aplicações clientes do servidor de banco de dados continuaram funcionando normalmente, e as operações de gravação foram realizadas pelo servidor principal do outro fragmento, de maneira transparente para o usuário. Após a interrupção do servidor principal de um dos fragmentos, um dos servidores secundários é escolhido servidor principal e, com isso, a arquitetura é reorganizada rapidamente para funcionar com 2 servidores principais, distribuídos em 2 fragmentos.

Nesta implementação fragmentada da arquitetura, é possível dizer que a tolerância a falhas acontece até a interrupção do funcionamento de 3 máquinas, independentemente do fragmento atingido. Assim, a partir do desligamento da quarta máquina dos fragmentos do *cluster*, o servidor de banco de dados fica indisponível.

### 4.2.3 Análise dos Resultados dos Testes das Implementações da Arquitetura

Os testes de inserção de dados realizados na arquitetura de armazenamento do Comune apresentaram resultados interessantes. O banco de dados MongoDB (modo *standalone*) se destacou, apresentando melhor média de tempo de inserção de dados em todos os testes: armazenamento de dados convencionais, multimídia e geográficos. Apesar de apresentar melhores resultados no tempo de armazenamento de dados, essa forma de implementação da arquitetura não suporta tolerância a falhas, o que pode comprometer a disponibilidade do SIGV.

A implementação da arquitetura que utiliza o banco de dados MongoDB replicado apresenta tempos de inserção de dados próximos aos registrados pelo MongoDB com um único servidor. Em alguns testes realizados é possível visualizar um melhor desempenho desta implementação, como foi o caso da inserção de 1.000 e 10.000 documentos de dados convencionais; e de 1.000, 10.000 e 100.000 documentos de dados geográficos. A grande vantagem da arquitetura replicada em relação a arquitetura de banco de dados único no MongoDB, é que, neste caso, tem-se tolerância a falhas a partir da cópia e sincronização dos dados do servidor de banco de dados. Em caso de falha de um dos servidores de banco de dados, o funcionamento (disponibilidade) do sistema não é comprometido.

A implementação da arquitetura utilizando o banco de dados MongoDB fragmentado apresentou bom desempenho, principalmente, nos testes de inserção de dados convencionais e geográficos. Essa implementação apresenta-se como uma alternativa mais viável para atender um SIGV que necessite armazenar grandes quantidades de dados, e apresentar alta disponibilidade. Apesar de apresentar tempos médios de inserção de dados pouco maiores que a implementação do MongoDB replicado, pode-se dizer que esta alternativa implementa a escalabilidade horizontal de fato, o que pode ser considerada uma vantagem em relação a arquitetura replicada.

A implementação da arquitetura que utilizou o banco de dados CouchDB apresentou desempenho inferior na inserção de dados em, praticamente, todos os testes realizados. O CouchDB apresentou um melhor desempenho somente na inserção de 100.000 e 500.000 documentos de dados multimídia em relação ao PostgreSQL. É importante ressaltar que o formato dos documentos armazenados no CouchDB e MongoDB são exatamente os mesmos, porém, esses bancos de dados apresentaram resultados bem diferentes em relação ao tempo de inserção de dados. O CouchDB foi escolhido para ser comparado aos outros bancos de dados nas implementações da arquitetura, por ser um dos bancos de dados baseado em documento mais utilizados atualmente, e por possuir suporte ao armazenamento de dados espaciais [30].

A implementação da arquitetura que utilizou o PostgreSQL apresentou em média tempos de inserção de dados melhores que a implementação utilizando o banco de dados CouchDB. No entanto, as implementações que utilizaram o banco de dados MongoDB apresentaram melhor desempenho, lidam melhor com dados heterogêneos e são facilmente escaláveis. Neste caso, é possível concluir que a arquitetura de armazenamento de dados para Sistemas de Informação Geográfica Voluntária utilizando banco de dados NoSQL baseado em documento é totalmente viável e interessante para atender os requisitos de escalabilidade e heterogeneidade, que são considerados essenciais para aplicações dessa natureza.

Não se encontrou na literatura trabalhos que definem implementações de SIGV com arquiteturas de armazenamento de dados que utilizam bancos de dados NoSQL. Assim, citam-se alguns diferenciais e algumas contribuições dessa arquitetura em comparação a outros trabalhos relacionados:

- Flexibilidade em relação a capacidade de armazenamento e crescimento do sistema de acordo com as necessidades.
- Implementação de níveis de confiabilidade através da replicação e fragmentação, que são importantes para sistemas de alta disponibilidade.
- Suporte ao armazenamento de dados de variados formatos.



- Especificação de uma arquitetura especializada para SIGV.
- Interoperabilidade da arquitetura, que permite que a aplicação cliente do SIGV seja escrita em qualquer linguagem de programação e execute sob qualquer plataforma, através da adoção de *web services* como componentes intermediários na comunicação entre a aplicação cliente do SIGV e o banco de dados.

# Capítulo 5

## Conclusão

A especificação de uma arquitetura de armazenamento de dados para SIGV apresenta diversos desafios, como armazenar e manipular grandes quantidades de dados, que, geralmente, estão estruturados em variados formatos. Neste contexto, definiu-se que os requisitos de escalabilidade e heterogeneidade devem ser considerados primordiais no projeto deste tipo de arquitetura.

Para atender aos requisitos de escalabilidade e heterogeneidade foi proposta uma arquitetura de armazenamento de dados utilizando banco de dados NoSQL. Os bancos de dados NoSQL possuem esquema de dados flexível, suportam dados de variados formatos, e possuem escalabilidade horizontal. Portanto, os NoSQL podem ser considerados uma boa alternativa para o armazenamento de dados em SIGV.

Realizando a pesquisa bibliográfica e comparando alguns tipos de bancos de dados NoSQL notou-se que os bancos de dados baseados em documento possuem uma maior flexibilidade no armazenamento de dados heterogêneos, e podem armazenar arquivos sem conhecer previamente a estrutura do documento a ser armazenado. Além disso, os bancos de dados baseados em documento possuem bom suporte ao armazenamento de dados geográficos, que são um tipo de dados que devem ser tratados pelos SIGV.

Na categoria de bancos de dados NoSQL baseados em documento, a adoção do BD NoSQL MongoDB mostrou-se a melhor opção, pois sua tecnologia suporta características consideradas essenciais para os SIGV, como alta variabilidade dos formatos de dados; boa consistência; simplicidade na arquitetura de replicação e fragmentação de dados, que permite que o sistema adapte-se para trabalhar com muitos usuários e com grande quantidade de dados; além de ter suporte a indexação e armazenamento de dados geográficos. Além disso, os testes realizados nas implementações da arquitetura demonstraram que o banco de dados MongoDB apresentou melhor desempenho em relação aos outros bancos de dados comparados: PostgreSQL e CouchDB.

Esta arquitetura foi validada por duas provas de conceito, que podem funcionar de

forma eficiente em uma aplicação real. Os conhecimentos aqui adquiridos podem ser aplicados em diversos tipos de arquitetura de banco de dados e irão ampliar as possibilidades de aplicação desse estudo. Os resultados da primeira prova de conceito foram publicados na Conferência Ibérica de Sistemas e Tecnologias da Informação – CISTI 2016 – com o título "*Voluntary Geographic Information Systems with Document-based NoSQL Databases*", que apresentou uma arquitetura alternativa de armazenamento de dados voltada para aplicações em SIGV.

Esta arquitetura de armazenamento de dados foi implementada utilizando três estratégias diferentes: único servidor, servidor replicado e servidor fragmentado. Todas as estratégias tem suas particularidades e usos. Nos testes realizados, a solução utilizando um único servidor apresentou melhor desempenho, porém, é importante ressaltar que as abordagens utilizando servidor replicado e fragmentado são interessantes para obtermos os benefícios da escalabilidade, disponibilidade e tolerância a falhas, que fazem parte dos requisitos dessa arquitetura.

Para trabalhos futuros, espera-se desenvolver e expandir este projeto. A lista a seguir apresenta algumas ideias e sugestões para a evolução do trabalho:

- Criar testes de leitura para comparação do desempenho dos bancos de dados PostgreSQL, CouchDB, e MongoDB, utilizando o projeto Comune.
- Realizar os testes de inserção e leitura de dados com múltiplos usuários simultâneos.
- Realizar a replicação e fragmentação dos bancos de dados PostgreSQL e CouchDB para comparação das arquiteturas distribuídas dos bancos de dados para o projeto Comune.
- Adaptar essa arquitetura para trabalhar com outros tipos de banco de dados, como Cassandra.

# Referências

- [1] V. Abramova e J. Bernardino. Nosql databases: MongoDB vs cassandra. *13th International C\* Conference on Computer Science and Software Engineering (C3S2E)*, pages 14–22, 2013. 17, 21, 22, 23
- [2] M.G. Abrantes e R. Carapuca. Explicit representation of data that depend on topological relationships and control over data consistency. *European Conference and Exhibition on Geographical Information Systems – EGIS/MARI*, pages 878–887, 1994. 14
- [3] A. Alabri e J. Hunter. Enhancing the quality and trust of citizen science data. *IEEE Sixth International Conference on e-Science*, pages 81–88, 2010. 8
- [4] Paolo Arcaini, Gloria Bordogna, e Simone Sterlacchini. Flexible querying of volunteered geographic information for risk management. *8th conference of the European Society for Fuzzy Logic and Technology*, pages 281–288, 2013. 38
- [5] Stan Aronoff. *Geographic Information Systems: A management perspective*. Editora Wdl Pubns, 1991. 5
- [6] C.S. Baptista, O.F. Lima Júnior, M.G. Oliveira, F.G. Andrade, T.E. Silva, e C.E.S. Pires. Using OGC services to interoperate spatial data stored in SQL and NoSQL databases. *XII Brazilian Symposium on Geoinformatics*, pages 61–72, 2011. 30, 35
- [7] K. A. Borges, C. A. Davis Júnior, e A. H. Laender. Omt-g: An object oriented data model for geographic applications. *GeoInformatica*, 5(3):221–260, 2001. 14
- [8] Eric Brewer. Cap twelve years later: How the rules have changed. *Computer*, 45(2):23–29, 2012. 17
- [9] Y. Bédard, C. Caron, Z. Maamar, B. Moulin, , e D. Vallière. Adapting data models for the design of spatio-temporal databases. *Computers, Environment and Urban Systems*, 20(1):19–41, 1996. 14
- [10] G. Câmara, Clodoveu Davis Júnior, e Antônio Miguel Vieira Monteiro. *Introdução a Ciência da Geoinformação*. INPE, 2005. 12, 14
- [11] Breno D.C. Camargos, Maristela Holanda, e Aletéia Araújo. A mobile public participation geographic information system architecture for collecting opinions about public services. *10th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6, 2015. xi, 33, 34, 36, 49, 51, 59

- [12] Marco A. Casanova, Gilberto Câmara, Clodoveu Davis Júnior, Lúbia Vinhas, e Gilberto Ribeiro De Queiroz. *Banco de Dados Geográficos*. MundoGEO, Curitiba, 2005. xi, 5, 6, 7, 12, 13, 14, 15
- [13] W.T. Castelein, L. Grus, J.W.H. Compvorts, e A.K. Bregt. A characterization of volunteered geographic information. *13th AGILE International Conference on Geographic Information Science*, pages 1–10, 2010. 1, 2, 8
- [14] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, e Robert E. Gruber. Bigtable: A distributed storage system for structured data. *7th USENIX Symposium on Operating Systems Design and Implementation*, 7:15–28, 2006. 15
- [15] Kristina Chodorow. *MongoDB: The definitive guide*. O’Reilly Media, Inc., 2013. 18, 23
- [16] David J. Cowen. *GIS versus CAD versus DBMS: What Are the Differences?* Taylor and Francis, Londres, 1988. 5
- [17] M. Craglia, M.F. Goodchild, A. Annoni, G. Câmara, M. Gould, W. Kuhn, D. Mark, I. Masser, D. Maguire, S. Liangm, e E. Parsons. Next-generation digital earth. *International Journal of Spatial Data Infrastructures Research*, 3:146–167, 2008. 8
- [18] C.A. Davis Júnior, H.S. Vellozo, e M.B. Pinheiro. A framework for web and mobile volunteered geographic information applications. *XIV Brazilian Symposium on Geoinformatics*, pages 147–157, 2013. 32, 33, 36, 38
- [19] William Gomes De Branco. *Comune - aplicativo android para questionários e coleta de relatos de usuários de serviços públicos*. Bachelor Thesis, Universidade de Brasília, 2016. xii, 49, 59, 60
- [20] G. De Candia e et al. Dynamo: Amazons highly available key-value store. *21th ACM SIGOPS symposium on Operating systems principles*, pages 205–220, 2007. 15, 21
- [21] Laura Diaz, Carlos Granell, Michael Gould, e Joaquin Huerta. Managing user-generated information in geospatial cyberinfrastructures. *Future Generation Computer Systems*, 27(3):304–314, 2011. 38
- [22] Ramez Elmasri e Shamkant B. Navathe. *Fundamentals of Database Systems*. Pearson, 7 edition, 2015. 14, 18, 20, 23, 24
- [23] Sarah Elwood. Volunteered geographic information: future research directions motivated by critical, participatory, and feminist gis. *GeoJournal*, 72(3–4):173–183, 2008. 1, 9
- [24] Andrew J. Flanagin e Miriam J. Metzger. The credibility of volunteered geographic information. *GeoJournal*, 72(3–4):137–148, 2008. 8
- [25] Adam Fowler. *NoSQL For Dummies*. John Wiley & Sons Inc., New Jersey, 2015. 16, 23

- [26] Michael F. Goodchild. Citizens as sensors: the world of volunteered geography. *GeoJournal*, 69(4):211–221, 2007. 1, 7
- [27] Michael F. Goodchild. Citizens as voluntary sensors: spatial data infrastructure in the world of web 2.0. *International Journal of Spatial Data Infrastructures Research*, 2:24–32, 2007. 8
- [28] Michael F. Goodchild e Linna Li. Assuring the quality of volunteered geographic information. *Spatial Statistics*, 1:110–120, 2012. 8
- [29] Jing Han, E. Haihong, Guan Le, e Jian Du. Survey on nosql database. *Pervasive Computing and Applications (ICPCA)*, pages 363–366, 2011. 17
- [30] Victoria Holt, Magnus Ramage, Karen Kear, e Nick Heap. The usage of best practices and procedures in the database community. *Information Systems*, 49:163–181, 2015. 72
- [31] M. Indrawan-Santiago. Database research: Are we at a crossroad? : Reflection on nosql. *15th Conference Network-Based Information Systems (NBiS)*, pages 45–51, 2012. 16, 20
- [32] C. Jardak, P. Mahonen, e J. Riihijarvi. Spatial big data and wireless networks: experiences, applications, and research challenges. *IEEE Network*, 28(4):26–31, 2014. 31, 35
- [33] G. Kusters, B.U. Pagel, e H.W. Six. Gis-application development with geoooa. *International Journal of Geographical Information Science*, 11(4):307–335, 1997. 14
- [34] Jae-Gil Lee e Minseo Kang. Geospatial big data: Challenges and opportunities. *Big Data Research*, 2(2):74–81, 2015. 12
- [35] Yan Li, GyoungBae Kim, Longri Wen, e Haeyoung Bae. MHB-Tree: A distributed spatial index method for document based nosql database system. *Ubiquitous Information Technologies and Applications*, 214:489–497, 2013. 31, 35
- [36] Luís E. O. Lizardo, Mirella M. Moro, e Clodoveu A. Davis Júnior. GeoNoSQL: Banco de dados geoespacial em NoSQL. *Computer on the Beach*, pages 303–309, 2014. 31, 35
- [37] Yan Ma, Haiping Wu, Lizhe Wang, Bormin Huang, Rajiv Ranjan, Albert Zomaya, e Wei Jie. Remote sensing big data computing: Challenges and opportunities. *Future Generation Computer Systems*, 51:47–60, 2014. 2, 18, 19, 31, 40
- [38] David J. Maguire, Michael F. Goodchild, e David W. Rhind. *Geographical Information Systems: Principles and Applications*. Longman, Londres, 1991. 6
- [39] MongoDB Manual. Mongoddb documentation. <https://docs.mongodb.com/manual/>, 2015. 18
- [40] Ian Masser. *Building European Spatial Data Infrastructures*. Esri Press, 2007. 6, 9

- [41] Dan McCreary e Ann Kelly. *Making Sense of NoSQL: A guide for managers and the rest of us*. Manning Publications Co., Shelter Island, NY, 2014. 16, 21, 22
- [42] Sam Meek, Mike J. Jackson, e Didier G. Leibovici. A flexible framework for assessing the quality of crowdsourced data. *AGILE International Conference on Geographic Information Science*, pages 1–7, 2014. 8
- [43] M. Miller, D. Medak, e O. Dravzen. Two-tier architecture for web mapping with NoSQL database CouchDB. *Geoinformatics Forum*, pages 62–71, 2011. 30
- [44] T.S. Miranda, J. Lisboa Filho, W.D. De Souza, O.C. Da Silva, e C.A. Davis Júnior. Volunteered geographic information in the context of local spatial data infrastructures. *Urban and Regional Data Management (UDMS)*, pages 123–138, 2011. 9, 32, 36, 38
- [45] A.B.M. Moniruzzaman e Syed Akhter Hossain. Nosql database: New era of databases for big data analytics - classification, characteristics and comparison. *International Journal of Database Theory and Application*, 6(4):1–14, 2013. 15, 22
- [46] S. Nativi, P. Mazzetti, Mattia Santoro, F. Papeschi, Max Craglia, e Osamu Ochiai. Big data challenges in building the global earth observation system of systems. *Environmental Modelling and Software*, 68:1–26, 2015. 1, 16
- [47] Pascal Neis e Dennis Zielstra. Recent developments and future trends in volunteered geographic information research: The case of openstreetmap. *Future Internet*, 6(1):76–106, 2014. 8, 9
- [48] J. L. Oliveira, F. Pires, e C. B. Medeiros. An environment for modeling and design of geographic applications. *GeoInformatica*, 1(1):29–58, 1997. 14
- [49] K. Orend. Analysis and classification of nosql databases and evaluation of their ability to replace an object-relational persistence layer. *Master Thesis, Technical University of Munich, Alemanha*, 2010. 26
- [50] Elaheh Pourabbas. *Geographical Information Systems: Trends and Technologies*. Taylor and Francis Group, Boca Raton, FL, USA, 2014. 19, 31, 35, 39
- [51] Philippe Rigaux, Michel Scholl, e Agnes Voisard. *Spatial databases: with application to GIS*. Morgan Kaufmann, 2001. 12
- [52] Pramod J. Sadalage e Martin Fowler. *NoSQL Essencial: Um guia conciso para o Mundo emergente da persistência poliglota*. Novatec, São Paulo, 2013. 1, 16, 18, 19, 20, 21, 22, 26, 61
- [53] S. Shekhar, M. Coyle, B. Goyal, D.R. Liu, e S. Sarkar. Data models in geographic information systems. *Communications of the ACM*, 40(4):103–111, 1997. 14
- [54] S.A. Sheppard. Wq: A modular framework for colleting, storing and utilizing experimental vgi. *1st ACM SIGSPATIAL International Workshop on Crowdsourced and Volunteered Geographic Information*, pages 62–69, 2012. 32, 36, 38

- [55] A. Silberstein, Chen Jianjun, D. Lomax, B. McMillan, e et al. Pnuts in flight: Web-scale data serving at yahoo. *Internet Computing*, 16(1):13–23, 2012. 16
- [56] Rosângela Silva. Bancos de dados geográficos: uma análise das arquiteturas dual (spring) e integrada (oracle spatial). *PhD Thesis, Universidade de São Paulo (USP)*, 2002. 14
- [57] Gonçalo da Cruz P. Sousa. Document-based databases: Estudo comparativo no âmbito das bases de dados nosql. *Master Thesis, Universidade do Minho, Portugal*, 2015. 18, 19
- [58] B.G. Tudorica e C. Bucur. A comparison between several nosql databases with comments and notes. *10th Roedunet International Conference (RoEduNet)*, pages 1–5, 2011. 15, 16
- [59] Gaurav Vaish. *Getting Started with NoSQL: Your guide to the world and technology of NoSQL*. Packt Publishing, Birmingham, 2013. 1
- [60] Harley Vera, Wagner Boaventura Filho, Maristela Holanda, e Aletéia Araújo. Geographic data modeling for nosql document-oriented databases. *GEOProcessing*, pages 63–68, 2015. xi, 24, 25, 51, 52, 60
- [61] Claudia Vitolo, Yehia Elkhatib, Dominik Reusser, Christopher J.A. Macleod, e Wouter Buytaert. Web technologies for environmental big data. *Environmental Modelling & Software*, 63:185–198, 2015. 1, 2, 18, 30, 35
- [62] X. Zhang, W. Song, e L. Liu. An implementation approach to store gis spatial data on nosql database. *Geoinformatics*, pages 1–5, 2014. 1, 23, 30, 35



# Apêndice A

## Estrutura dos documentos do banco de dados Comune (modelo não relacional)

---

```
1 {
2   "id": <objeto ID>,
3   "password": "<caractere>",
4   "firstName": "<caractere>",
5   "lastName": "<caractere>",
6   "dateOfBirth": "<data>",
7   "email": "<caractere>",
8   "phoneNumber": "<caractere>",
9   "mobileNumber": "<caractere>",
10  "registered_at": "<data-hora>",
11  "last_logged_in_at": "<data-hora>"
12 }
```

---

Código A.1: Estrutura do documento User do BD Comune.

---

```
1 {
2   "id": <objeto ID>,
3   "name": "<caractere>",
4   "nickname": "<caractere>",
5   "abbrevName": "<caractere>",
6   "address": "<caractere>",
7   "rating": <numero>,
8   "status": <numero>,
9   "location": {
10     "type": "<tipo objeto geografico>",
11     "coordinates": [<longitude>, <latitude>]
12   },
```

```

13     "category": <numero>,
14     "descCategory": "<caractere>",
15     "subcategory": <numero>,
16     "descSubCategory": "<caractere>",
17     "surveys": [{
18         "idSurvey": <numero>,
19         "openingTime": "<data-hora>",
20         "closingTime": "<data-hora>",
21         "active": <boolean>
22     }],
23     "workingDays": [{
24         "dayOfTheWeek": <numero>,
25         "openingTime": "<hora>",
26         "closingTime": "<hora>"
27     }, {
28         "dayOfTheWeek": <numero>,
29         "openingTime": "<hora>",
30         "closingTime": "<hora>"
31     }]
32 }

```

---

Código A.2: Estrutura do documento Place do BD Comune.

---

```

1  {
2     "id": <objeto ID>,
3     "idUser": <numero>,
4     "idPlace": <numero>,
5     "comment": "<caractere>",
6     "received_at": "<data-hora>",
7     "made_at": "<data-hora>",
8     "responses": [{
9         "idUser": <numero>,
10        "id": <numero>,
11        "comment": "<caractere>",
12        "received_at": "<data-hora>",
13        "made_at": "<data-hora>",
14        "visualized": <boolean>
15    }, {
16        "idUser": <numero>,
17        "id": <numero>,
18        "comment": "<caractere>",
19        "received_at": "<data-hora>",
20        "made_at": "<data-hora>",
21        "visualized": <boolean>
22    }]

```

23 }

---

Código A.3: Estrutura do documento Report do BD Comune.

---

```
1 {
2   "id": <objeto ID>,
3   "description": "<caractere>",
4   "created_at": "<data-hora>",
5   "updated_at": "<data-hora>",
6   "category": <numero>,
7   "descCategory": "<caractere>",
8   "subcategory": <numero>,
9   "descSubCategory": "<caractere>",
10  "questions": [{
11    "idQuestion": <numero>,
12    "description": "<caractere>",
13    "mandatory": <boolean>,
14    "type": "<caractere>",
15    "options": [{
16      "idOption": <numero>,
17      "descOption": "<caractere>"
18    }, {
19      "idOption": <numero>,
20      "descOption": "<caractere>"
21    }]
22  }, {
23    "idQuestion": <numero>,
24    "description": "<caractere>",
25    "mandatory": <boolean>,
26    "type": "<caractere>",
27    "options": [<caractere>, <caractere>]
28  }, {
29    "idQuestion": <numero>,
30    "description": "<caractere>",
31    "mandatory": <boolean>,
32    "type": "<caractere>",
33    "options": [<numero>, <numero>, <numero>, <numero>]
34  }]
35 }
```

---

Código A.4: Estrutura do documento Survey do BD Comune.

---

```
1 {
2   "id": <objeto ID>,
3   "idSurvey": <numero>,
4   "idPlace": <numero>,
```

```

5  "idUser": <numero>,
6  "started_at": "<data-hora>",
7  "completed_at": "<data-hora>",
8  "questions_answers": [{
9      "idQuestion": <numero>,
10     "answer": <numero>
11 }, {
12     "idQuestion": <numero>,
13     "answer": <caractere>
14 }, {
15     "idQuestion": <numero>,
16     "answer": <numero>
17 }]
18 }

```

---

Código A.5: Estrutura do documento SurveyAnswer do BD Comune.

```

1  {
2  "idReport": <numero>,
3  "idPlace": <numero>,
4  "idUser": <numero>,
5  "Content-Type": "<caractere>",
6  "filename": "<caractere>",
7  "extension": "<caractere>",
8  "uploadDate": "<data-hora>",
9  "length": <numero>,
10 "chunkSize": <numero>,
11 "md5": "<caractere>"
12 }

```

---

Código A.6: Estrutura do documento Fs.Files do BD Comune.

# Apêndice B

## *Web services* implementados na arquitetura do projeto Comune

### *Web services* relacionados aos usuários

- users/authenticateUser: Serviço para autenticação de usuários.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: E-mail e senha do usuário.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com informações básicas do usuário. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
- users/registerUser: Serviço para criação de usuário.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Nome, e-mail, senha, telefone, data de nascimento e foto do usuário.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com código de identificação do usuário criado. Se o e-mail informado na entrada já estiver cadastrado, exibe uma mensagem informando o erro e retorna um status de conflito. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
- users/uploadUserPhoto: Serviço para armazenar a foto do usuário.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST

- Entrada: Código de identificação do usuário e arquivo de imagem.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com código de identificação da imagem armazenada. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
- users/downloadFile: Serviço para retornar um arquivo binário.
    - Acesso via: Requisição HTTP
    - Método de envio de dados: GET
    - Entrada: Código de identificação do arquivo binário.
    - Retorno: Em caso de sucesso, retorna o conjunto de bytes do arquivo binário. Em caso de falha, retorna um conjunto de bytes vazio.
  - users/getUserPhotos: Serviço para retornar as fotos de um usuário.
    - Acesso via: Requisição HTTP
    - Método de envio de dados: POST
    - Entrada: Código de identificação do usuário.
    - Retorno: Em caso de sucesso, retorna uma lista das fotos do usuário. Em caso de falha, retorna um objeto JSON vazio.

### **Web services relacionados aos estabelecimentos públicos**

- places/registerNewPlace: Serviço para cadastro de estabelecimento público.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Nome, apelido, nome abreviado, endereço, longitude, latitude, categoria do estabelecimento.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com código de identificação, longitude e latitude do estabelecimento público criado. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
  - Observações: Este serviço armazena dados convencionais e utiliza um objeto geométrico ponto, adotando a notação GeoJSON e as coordenadas de longitude e latitude, para armazenar a localização do estabelecimento público.
- places/addSurveyToPlace: Serviço para adicionar uma pesquisa a um estabelecimento público.

- Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação do estabelecimento público, código de identificação da pesquisa, data/hora de início e término da pesquisa.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com uma mensagem de confirmação. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
- places/getNearbyPlaces: Serviço para retornar uma lista de estabelecimentos públicos próximos a determinado raio de distância.
    - Acesso via: Requisição HTTP
    - Método de envio de dados: GET
    - Entrada: Longitude, Latitude, e raio de distância a ser pesquisado.
    - Retorno: Caso exista estabelecimentos públicos próximos, retorna um objeto JSON com uma lista dos estabelecimentos. Caso não exista estabelecimentos públicos próximos, retorna um objeto JSON vazio.
    - Observações: Este serviço realiza uma consulta espacial no banco de dados para retornar os estabelecimentos públicos próximos às coordenadas geográficas do aparelho celular do usuário voluntário.
- places/getPlaces: Serviço para retornar a lista de estabelecimentos públicos cadastrados.
    - Acesso via: Requisição HTTP
    - Método de envio de dados: GET
    - Entrada: Não há.
    - Retorno: Caso exista estabelecimentos públicos cadastrados, retorna um objeto JSON com a lista dos estabelecimentos públicos cadastrados. Caso não exista estabelecimentos públicos cadastrados, retorna um objeto JSON vazio.
- places/getPlaceById: Serviço para retornar o estabelecimento público especificado pelo código de identificação.
    - Acesso via: Requisição HTTP
    - Método de envio de dados: GET
    - Entrada: Código de identificação do estabelecimento público.

- Retorno: Caso o estabelecimento público seja encontrado, retorna um objeto JSON com as informações do estabelecimento público. Caso o estabelecimento público não seja encontrado, retorna um objeto JSON vazio.
- places/findNearbyPlaces: Serviço para retornar uma lista de estabelecimentos públicos próximos a determinado raio de distância e que pertençam a determinada categoria.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: GET
  - Entrada: Longitude, Latitude, raio de distância e categoria.
  - Retorno: Caso exista estabelecimentos públicos que atendam os critérios estabelecidos, retorna um objeto JSON com uma lista dos estabelecimentos. Caso não exista estabelecimentos públicos que atendam os critérios estabelecidos, retorna um objeto JSON vazio.
  - Observações: Este serviço realiza uma consulta espacial no banco de dados para retornar os estabelecimentos públicos próximos às coordenadas geográficas do aparelho celular do usuário voluntário.
- places/getPlacesReportedByUser: Serviço para retornar a lista de estabelecimentos públicos que receberam relatos de determinado usuário.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação do usuário.
  - Retorno: Caso exista estabelecimentos públicos que receberam relatos do usuário especificado, retorna um objeto JSON com uma lista dos estabelecimentos. Caso não exista estabelecimentos públicos que receberam relatos do usuário especificado, retorna um objeto JSON vazio.
- places/uploadPlacePhoto: Serviço para armazenar a foto do estabelecimento público.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação do estabelecimento público e arquivo de imagem.



- Retorno: Em caso de sucesso, retorna um objeto JSON com código de identificação da imagem armazenada. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
- places/getPlacePhotos: Serviço para retornar as fotos de um estabelecimento público.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação do estabelecimento público.
  - Retorno: Em caso de sucesso, retorna uma lista das fotos do estabelecimento público. Em caso de falha, retorna um objeto JSON vazio.

### **Web services relacionados às Pesquisas**

- surveys/registerNewSurvey: Serviço para cadastrar pesquisa.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Descrição e categoria da pesquisa.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com código de identificação da pesquisa criada. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
- surveys/updateSurveyById: Serviço para atualizar uma pesquisa a partir de determinado código de identificação.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação da pesquisa, descrição e categoria da pesquisa.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com uma mensagem de confirmação. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
- surveys/registerNewQuestionToSurvey: Serviço para adicionar uma questão a uma determinada pesquisa.
  - Acesso via: Requisição HTTP

- Método de envio de dados: POST
  - Entrada: Código de identificação da pesquisa, descrição, tipo e opções da questão.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com uma mensagem de confirmação. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
- `surveys/getSurveyById`: Serviço para retornar uma determinada pesquisa pelo código de identificação.
    - Acesso via: Requisição HTTP
    - Método de envio de dados: POST
    - Entrada: Código de identificação da pesquisa.
    - Retorno: Caso exista a pesquisa especificada, retorna um objeto JSON com as informações da pesquisa. Caso não exista a pesquisa especificada, retorna um objeto JSON vazio.
- `surveys/saveSurveyAnswers`: Serviço para armazenar as respostas da pesquisa.
    - Acesso via: Requisição HTTP
    - Método de envio de dados: POST
    - Entrada: Código de identificação da pesquisa, código de identificação do estabelecimento público, código de identificação do usuário, data de início do preenchimento, e respostas das questões da pesquisa.
    - Retorno: Em caso de sucesso, retorna um objeto JSON com código de identificação da pesquisa. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
- `surveys/getAvailableSurveys`: Serviço para listar as pesquisas disponíveis em um determinado estabelecimento público.
    - Acesso via: Requisição HTTP
    - Método de envio de dados: POST
    - Entrada: Código de identificação do estabelecimento público.
    - Retorno: Caso exista pesquisas disponíveis para o estabelecimento público especificado, retorna um objeto JSON com as informações das pesquisas. Caso não exista pesquisas disponíveis para o estabelecimento público especificado, retorna um objeto JSON vazio.

## **Web services relacionados aos Relatos**

- reports/saveNewReport: Serviço para cadastrar um relato para um determinado estabelecimento público.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação do usuário, código de identificação do estabelecimento público e comentário do relato.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com código de identificação do relato criado. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
  
- reports/uploadReportPicture: Serviço para armazenar uma imagem do relato para o estabelecimento público.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação do relato e arquivo de imagem.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com código de identificação da imagem armazenada. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
  
- reports/uploadReportFootage: Serviço para armazenar um vídeo do relato para o estabelecimento público.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação do relato e arquivo de vídeo.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com código de identificação do vídeo armazenado. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
  
- reports/uploadReportAudio: Serviço para armazenar um áudio do relato para o estabelecimento público.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST

- Entrada: Código de identificação do relato e arquivo de áudio.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com código de identificação do áudio armazenado. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
- reports/saveNewResponseForReport: Serviço para armazenar comentário de um determinado relato.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação do relato, código de identificação do usuário, e comentários do relato.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com as informações do comentário do relato. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
- reports/markReportResponseAsVisualized: Serviço para marcar a resposta do relato como visualizada.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação da resposta do relato.
  - Retorno: Em caso de sucesso, retorna um objeto JSON com uma mensagem de confirmação. Em caso de falha, retorna um objeto JSON com a descrição e o status do erro.
- reports/getReportById: Serviço para retornar um determinado relato pelo código de identificação.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação do relato.
  - Retorno: Caso exista o relato especificado, retorna um objeto JSON com as informações do relato. Caso não exista o relato especificado, retorna um objeto JSON vazio.
- reports/getReportResponsesById: Serviço para retornar uma lista de respostas de um determinado relato pelo código de identificação.

- Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação do relato.
  - Retorno: Caso exista respostas do relato especificado, retorna um objeto JSON com uma lista de respostas do relato. Caso não exista respostas do relato especificado, retorna um objeto JSON vazio.
- reports/getReportResponseById: Serviço para retornar uma determinada resposta através do código de identificação.
    - Acesso via: Requisição HTTP
    - Método de envio de dados: POST
    - Entrada: Código de identificação da resposta do relato.
    - Retorno: Caso exista a resposta especificada, retorna um objeto JSON com as informações da resposta do relato. Caso não exista a resposta especificada, retorna um objeto JSON vazio.
- reports/getPlaceReportsSubmittedByUser: Serviço para retornar uma lista de relatos enviados por um determinado usuário em um estabelecimento público específico.
    - Acesso via: Requisição HTTP
    - Método de envio de dados: POST
    - Entrada: Código de identificação do usuário e código de identificação do estabelecimento público.
    - Retorno: Caso exista relatos enviados pelo usuário especificado, retorna um objeto JSON com uma lista de relatos. Caso não exista relatos enviados pelo usuário especificado, retorna um objeto JSON vazio.
- reports/getNewResponsesForUserReports: Serviço para retornar uma lista de respostas de relatos ainda não visualizadas por um determinado usuário.
    - Acesso via: Requisição HTTP
    - Método de envio de dados: POST
    - Entrada: Código de identificação do usuário.
    - Retorno: Caso exista respostas não visualizadas para o usuário especificado, retorna um objeto JSON com uma lista de respostas de relatos. Caso não exista respostas não visualizadas para o usuário especificado, retorna um objeto JSON vazio.

- reports/getReportPhotos: Serviço para retornar as fotos de um relato.
  - Acesso via: Requisição HTTP
  - Método de envio de dados: POST
  - Entrada: Código de identificação do relato.
  - Retorno: Em caso de sucesso, retorna uma lista das fotos do relato. Em caso de falha, retorna um objeto JSON vazio.