

UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

**ANÁLISE DE BANCOS DE DADOS NOSQL E
DESENVOLVIMENTO DE UMA APLICAÇÃO**

TRABALHO DE GRADUAÇÃO

Ivan Isaías Friess

Santa Maria, RS, Brasil

2013

ANÁLISE DE BANCOS DE DADOS NOSQL E DESENVOLVIMENTO DE UMA APLICAÇÃO

por

Ivan Isaías Friess

Trabalho de Graduação apresentado ao Curso de Ciência da Computação
da Universidade Federal de Santa Maria (UFSM, RS), como requisito
parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Prof^ª Dr^ª Deise de Brum Saccol

Trabalho de Graduação N^o 352
Santa Maria, RS, Brasil

2013

**Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação**

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

**ANÁLISE DE BANCOS DE DADOS NOSQL E
DESENVOLVIMENTO DE UMA APLICAÇÃO**

elaborado por
Ivan Isaías Friess

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Prof^a Dr^a Deise de Brum Saccol
(Presidente/Orientador)

Prof Dr Eduardo Kessler Piveta (UFSM)

Prof^a Dr^a Marcia Pasin (UFSM)

Santa Maria, 20 de fevereiro de 2013.

AGRADECIMENTOS

Gostaria de agradecer, em primeiro lugar, à minha família pelo incentivo de sempre. Em especial aos meus pais, pelo apoio, força e dedicação.

Agradeço à professora Deise pela orientação deste trabalho. À banca avaliadora, professor Eduardo e professora Marcia, pelas sugestões de melhorias.

Também agradeço ao Centro de Processamento de Dados (CPD) da UFSM pela chance de realizar um estágio e, em especial, ao colegas do setor de Divisão de Redes e Sistemas Básicos, pelos ensinamentos e o ótimo ambiente de trabalho.

Não poderia deixar de agradecer à Universidad Nacional del Este (UNE) pela oportunidade que me foi dada de realizar um intercâmbio, à Facultad Politécnica (FPUNE) pela excelente receptividade que tive. À todas as pessoas que pude conhecer e conviver durante o período de experiência no Paraguai.

Enfim, agradeço a todas as pessoas que, de alguma forma, contribuíram para que este momento tenha chegado.

*“Nossa maior fraqueza está em desistir.
O caminho mais certo de vencer é tentar mais uma vez.”*
— THOMAS EDISON

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

ANÁLISE DE BANCOS DE DADOS NOSQL E DESENVOLVIMENTO DE UMA APLICAÇÃO

Autor: Ivan Isaías Friess
Orientador: Prof^a Dr^a Deise de Brum Saccol
Local e data da defesa: Santa Maria, 20 de fevereiro de 2013.

Os bancos de dados NoSQL estão tornando-se cada vez mais conhecidos e utilizados nos últimos anos, em razão da necessidade de armazenamento de grande quantidade de dados com os quais as organizações têm que trabalhar e também do aumento de usuários dos sistemas. Os bancos de dados tradicionalmente utilizados, como os relacionais, apresentam fatores limitantes devido ao fato de não apresentarem muita flexibilidade na estruturação dos dados. Com isto, os Bancos de Dados NoSQL são uma alternativa que objetiva suprir estas e outras deficiências, como as questões de escalabilidade e disponibilidade do sistema.

No entanto, os Bancos de Dados NoSQL são recentes e ainda pouco conhecidos, o que pode ser um empecilho no momento de escolher um destes para gerenciamento de uma base de dados. Assim, o objetivo deste trabalho é realizar uma pesquisa, selecionando alguns bancos de dados NoSQL e compará-los entre si, mostrando as funcionalidades comuns e as particulares. Após a comparação, é escolhido um Banco de Dados NoSQL, de acordo com as características de funcionamento, para o desenvolvimento de uma aplicação, que utilizará o próprio banco de dados NoSQL para a realização de consultas e modificações de sua base de dados, proporcionando assim um melhor entendimento sobre esta tecnologia.

Palavras-chave: NoSQL, Banco de Dados.

ABSTRACT

Undergraduate Final Work
Undergraduate Program in Computer Science
Federal University of Santa Maria

NOSQL DATABASES ANALYSIS AND DEVELOPMENT OF AN APPLICATION

Author: Ivan Isaías Friess
Advisor: Prof^ª Dr^ª Deise de Brum Saccol

NoSQL databases are becoming more popular and more used in recent years, mainly due to the large amount of data that organizations have to deal with and to the increasing number of system users. Databases commonly used have limiting factors such as the lack of flexibility in structuring data. Thus, NoSQL databases are an alternative that aims to overcome deficiencies like scalability and availability.

However, NoSQL databases are recent and practically unknown; choose one of these management systems can be an obstacle for managing a database. The goal of this work is to perform a search by selecting and comparing some NoSQL databases, pointing out the common and the particular features. After the comparison, one NoSQL database is chosen according to some features, in order to develop an application that uses the NoSQL database to pose queries and updates over the data, allowing a better understanding of this technology.

Keywords: NoSQL, databases.

LISTA DE FIGURAS

Figura 2.1: Exemplo documento JSON (Wikipedia, 2012)	18
Figura 2.2: Arquitetura Redis (Campanelli, 2011).	20
Figura 2.3: Modelo de Cluster no Riak (Riak, 2012).	23
Figura 2.4: Cluster do Cassandra (Cassandra, 2012)	26
Figura 2.5: Modelo de dados do Apache Cassandra (Cho, 2010)	27
Figura 2.6: Repositório do Apache Cassandra	28
Figura 2.7: Chave Publica do repositório	28
Figura 2.8: Instalação do Apache Cassandra	28
Figura 2.9: Arquitetura HBase	31
Figura 2.10: Arquitetura CouchDB	33
Figura 2.11: Arquitetura Cluster MongoDB (Orend, 2010).	36
Figura 2.12: Modelo de dados no Neo4j (Neo4j, 2012).	40
Figura 3.1: Modelo documento dos metadados	45
Figura 3.2: Documento inserido na coleção "files", incluindo metadados	45
Figura 3.3: Modelo documento na coleção "chunks"	46
Figura 3.4: Diagrama de caso de uso da aplicação	46
Figura 4.1: Diagrama de classes da aplicação	50
Figura 4.2: Código do método conectar.....	51
Figura 4.3: Trecho do código do método formataConsulta	51
Figura 4.4: Código do método consultaBanco	52
Figura 4.5: Código do método excluir.....	52
Figura 4.6: Código do método insereDoc	53
Figura 4.7: Código do método editarMetadados	54
Figura 4.8: Tela principal Usuário Cliente	56
Figura 4.9: Tela de Autenticação como Administrador	56
Figura 4.10: Tela de Visualização dos Detalhes	57
Figura 4.11: Tela principal do Administrador	59
Figura 4.12: Tela de Inserção de Monografias	60
Figura 4.13: Documento gerado na inserção da monografia	61
Figura 4.14: Caixa de texto para consulta avançada	62
Figura 4.15: Tela de Edição de Monografias.	65

LISTA DE TABELAS

Tabela 2.1: Tabela comparativa NoSQL Databases.....	42
Tabela 4.1: Carga de inserções na base dados.....	67

LISTA DE APÊNDICES

APÊNDICE A – Código fonte da aplicação.....	74
A.1 – BuscaDocumento.java.....	74
A.2 – BuscaDocAdm.java.....	81
A.3 – NovoDocumento.java.....	88
A.4 – EditarDocumento.java.....	94
A.5 – VisualizarDocumento.java.....	97
A.6 – TableModelDoc.java.....	99
A.7 – Metadado.java.....	100

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
AGPL	<i>Affero General Public License</i>
API	<i>Application Programming Interface</i>
ASF	<i>Apache Software Foundation</i>
BASE	<i>Basically Available, Soft state, Eventual consistency</i>
BSON	<i>Binary JSON</i>
CQL	<i>Cypher Query Language</i>
HDFS	<i>Hadoop Distributed Filesystem</i>
HTTP	<i>Hypertext Markup Language</i>
JDK	<i>Java Development Kit</i>
JSON	<i>JavaScript Object Notation</i>
NOSQL	<i>Not Only Structured Query Language</i>
REST	<i>Representational State Transfer</i>
SGBD	Sistema de Gerenciamento de Banco de Dados
SGBDR	Sistema Gerenciador de Banco de Dados Relacional
SQL	<i>Structured Query Language</i>

SUMÁRIO

1 INTRODUÇÃO.....	13
1.1 JUSTIFICATIVA.....	14
1.2 OBJETIVOS.....	14
1.3 METODOLOGIA.....	15
1.4 ORGANIZAÇÃO DO TEXTO.....	15
2 REVISÃO DE LITERATURA.....	16
2.1 NOSQL.....	16
2.1.1 Chave/Valor.....	17
2.1.2 Orientado a colunas.....	17
2.1.3 Orientado a documentos.....	18
2.1.4 Baseado em Grafos.....	19
2.2 SGBDs NOSQL.....	19
2.2.1 Redis.....	19
2.2.2 Riak.....	23
2.2.3 Cassandra.....	25
2.2.4 HBase.....	30
2.2.5 CouchDB.....	32
2.2.6 MongoDB.....	35
2.2.7 Neo4j.....	39
2.3 COMPARATIVO ENTRE OS BANCOS DE DADOS NOSQL.....	41
3 PROPOSTA DO TRABALHO.....	44
3.1 PROPOSTA DE DESENVOLVIMENTO.....	44
3.2 DIAGRAMA DE CASOS DE USO.....	46
3.2.1 Especificação do caso de uso Consultar banco de dados.....	47
3.2.2 Especificação do caso de uso Visualizar metadados do documento.....	47
3.2.3 Especificação do caso de uso Inserir documento.....	48
3.2.4 Especificação do caso de uso Alterar documento.....	48
3.2.5 Especificação do caso de uso Remover documento.....	48
4 DESENVOLVIMENTO DA APLICAÇÃO.....	49
4.1 CLASSES IMPLEMENTADAS.....	49
4.1.1 Classe BuscaDocumento e Classe BuscaDocAdm.....	51
4.1.2 Classe NovoDocumento.....	53
4.1.3 Classe EditarDocumento.....	54
4.1.4 Classe VisualizarDetalhes.....	54
4.1.5 Classe TableModelDoc e Classe Metadado.....	54
4.2 UTILIZAÇÃO DA APLICAÇÃO.....	55
4.2.1 Utilização como Usuário Cliente.....	55
4.2.2 Utilização como usuário Administrador.....	58
4.2.3 Tratamento de erros.....	66
4.3 TESTES DE DESEMPENHO.....	67
4.4 DOWNLOAD DA APLICAÇÃO.....	68
5 CONCLUSÕES E TRABALHOS FUTUROS.....	69
REFERÊNCIAS.....	71
APÊNDICES.....	74

1 INTRODUÇÃO

Durante um longo período o modelo de dados relacional tem sido amplamente utilizado pela maioria dos Sistemas de Gerenciamento de Banco de Dados (SGBD). Ainda hoje é assim. Entretanto, nos últimos anos vem ocorrendo um crescimento significativo de aplicações, recursos *web* e tudo que se refere a sistemas computacionais. Essas aplicações geram grande volume de dados e têm o desafio de servir uma grande quantidade de usuários, que esperam o perfeito funcionamento dos sistemas. Nesse cenário, a utilização do modelo relacional não se mostra tão eficiente, uma vez que este modelo possui sérias limitações quanto à escalabilidade horizontal, já que não foi projetado inicialmente para tal. Como organizar os dados de um Sistema de Gerenciamento de Banco de Dados Relacional (SGBDR) em sistemas distribuídos é complexo, quase sempre se acaba recorrendo à escalabilidade vertical para aumento de desempenho, ou seja, faz-se *upgrade* dos sistemas de *hardware* do servidor. No entanto, a escalabilidade vertical tem limite. Para suprir as deficiências encontradas nos SGBDRs e, principalmente, pela necessidade de escalabilidade em bancos de dados, sem perda de desempenho, surgem alternativas ao modelo relacional. Uma delas, a que ganhou mais destaque, é o paradigma NoSQL (Orend, 2010).

O termo NoSQL foi utilizado pela primeira vez em 1998, por Carlo Strozzi, para nomear um SGBD de código aberto desenvolvido por ele e que não oferecia uma interface *Structured Query Language* (SQL), entretanto, ainda era baseado na arquitetura relacional. Era um sistema experimental e que não foi muito utilizado. O termo acabou caindo no esquecimento até o ano de 2009, quando ressurgiu em um evento promovido por Johan Oskarsson e Eric Evans. O evento teve como objetivo discutir o crescente surgimento de

soluções *open source* de armazenamento de dados distribuídos não relacionais. A partir de então o termo NoSQL foi redefinido para descrever as soluções de armazenamento de dados não relacionais, inclusive aquelas já existentes, como é o caso do BigTable do Google, lançado em 2004, e o Dynamo da Amazon, de 2007, ambas proprietárias. Após isso, outros projetos surgiram, como o Cassandra, o CouchDB, o MongoDB, etc.

Todas soluções apresentam características em comum e outras particulares. A fim de analisar as características dos bancos de dados NoSQL, será realizado um estudo em alguns bancos existentes e também será desenvolvida uma aplicação para um desses bancos.

1.1 JUSTIFICATIVA

Tem-se comentado muito sobre as vantagens de utilização de banco de dados NoSQL como alternativa aos bancos relacionais, principalmente em cenários onde faz-se necessário o armazenamento de grande volume de dados com alta disponibilidade e grande escalabilidade. Muitas empresas (Google, Facebook, Amazon, LinkedIn, Twitter, etc...) têm relatado sucesso no uso de banco de dados NoSQL, solucionando assim o problema de escalabilidade e desempenho.

No entanto, é preciso entender o contexto e dispor de uma análise prévia sobre tais bancos, de modo a escolher o que oferece as melhores funcionalidades e seja o mais apropriado para determinada aplicação.

Sendo assim, serão selecionados alguns bancos de dados NoSQL, *Open Source*, para a realização de um estudo e comparação entre eles. A seleção dos bancos privilegia os que estejam sendo mais utilizados no mercado, de modo a escolher um deles para o desenvolvimento de uma aplicação.

1.2 OBJETIVOS

O objetivo do trabalho é realizar uma pesquisa selecionando alguns Bancos de Dados NoSQL para estudo e comparação, selecionando um dos bancos estudados para o desenvolvimento de uma aplicação para manipular dados armazenados neste banco.

1.3 METODOLOGIA

A metodologia utilizada para atingir o objetivo do trabalho é a que segue:

- Pesquisa e seleção de *NoSQL Databases*;
- Estudo individual de cada banco de dados;
- Comparação dos Bancos de Dados e escolha de um para o desenvolvimento da aplicação;
- Definição da aplicação a ser desenvolvida;
- Desenvolvimento da aplicação;
- Testes da aplicação;
- Elaboração da parte escrita.

1.4 ORGANIZAÇÃO DO TEXTO

O texto encontra-se estruturado da seguinte forma:

No capítulo 2 são abordados os conceitos dos *NoSQL Databases*, as suas características e subdivisões. Além disso, são apresentados os bancos de dados *NoSQL* estudados neste trabalho e uma análise comparativa entre eles.

O capítulo 3 apresenta o Banco de Dados escolhido para realizar a implementação de uma aplicação, explicando quais motivos levaram à escolha deste. Também é exposta a proposta de aplicação desenvolvida, mostrando quais são as funcionalidades contidas na ferramenta. Um diagrama de casos de uso é usado para auxiliar o entendimento.

O capítulo 4 mostra como foi desenvolvida a aplicação proposta, com uma breve explicação sobre a aplicação. Também são comentadas quais as tecnologias utilizadas para o desenvolvimento e, ainda, são exibidas algumas imagens da ferramenta para auxiliar na explicação da utilização da aplicação.

O capítulo 5 apresenta o encerramento do trabalho, considerações e sugestões de melhorias para trabalhos futuros.

2 REVISÃO DE LITERATURA

Este capítulo tem por finalidade apresentar os conceitos de algumas tecnologias emergentes na Computação, como é o caso dos Bancos de Dados NoSQL e o paradigma BASE (*Basically Available, Soft state, Eventual consistency*) e, também, apresentar os Bancos de Dados NoSQL selecionados para análise neste trabalho.

2.1 NOSQL

Termo que descreve uma nova classe de banco de dados que vêm tendo um crescimento de uso acentuado nos últimos anos, principalmente em função de algumas limitações existentes nos bancos de dados relacionais. Apesar de o termo significar *Not Only SQL* (não apenas SQL), a principal característica desses bancos é de não ter comportamento relacional. Inclusive, não há nada que impeça o uso de linguagem SQL. A implicação do termo NoSQL é que os objetivos desses bancos de dados, modernos e distribuídos, são, na maioria das vezes, distintos dos objetivos dos bancos de dados relacionais.

Os bancos de dados NoSQL geralmente seguem os princípios do paradigma BASE, em detrimento das propriedades ACID, priorizando o desempenho e a disponibilidade. Esse paradigma fornece aos bancos de dados características como: a) Basicamente disponível (*Basically Available*), ou seja, o sistema parece estar funcionando o tempo todo; b) Estado leve (*Soft State*), ou seja, o sistema não precisa ser consistente o tempo todo; c) Consistência eventual (*Eventual consistency*), ou seja, o sistema torna-se consistente no momento devido (Pritchett, 2008).

Com NoSQL *Databases*, pode-se gerenciar grande quantidade de dados sem perda de desempenho. Isso é plausível porque esses bancos prezam pela inexistência de transações e cláusulas *join*. Ainda possuem configuração facilitada para escalabilidade horizontal e, na

maioria das vezes, suporte nativo à replicação dos dados. A escalabilidade horizontal corresponde à adição de nodos (máquinas servidoras), enquanto a escalabilidade vertical é o aumento de desempenho de apenas um servidor central. A escalabilidade vertical pode ter um limite de *hardware*, já na horizontal não há limites de nodos, porém é mais complicada de ser arquitetada e necessita de facilitadores. No caso, o uso de bancos de dados NoSQL é um facilitador.

Outra característica dos NoSQL *Databases* é o armazenamento de dados sem estrutura pré-definida, ou seja, não é preciso definir no momento de criação do banco os tipos de dados que serão armazenados; isso é feito dinamicamente, à medida que são inseridos.

Os bancos de dados NoSQL são classificados de acordo com a forma de armazenar os dados. Existem 4 tipos de categorias: chave/valor, orientado a colunas, orientado a documentos e baseados em grafos (Popescu, 2010).

2.1.1 Chave/Valor

O armazenamento do tipo chave/valor é semelhante ao uso de mapas ou de dicionários, onde os dados são endereçados por uma única chave, permitindo aos clientes colocar e solicitar valores por chaves. Esses sistemas podem conter dados estruturados ou não estruturados. São úteis para operações simples, que são baseadas somente em atributos chave. Por exemplo, sistemas com rápida troca de dados e com frequente escrita. Como a maioria dos armazenamentos chave/valor mantém seu conjunto de dados em memória, eles são bastante usados para *cache* de consultas SQL (Leavitt, 2010). Alguns bancos desta categoria são: Redis, Riak, Voldemort, MemcacheDB, etc. (FindTheBest, 2012).

2.1.2 Orientado a colunas

Um SGBD orientado a colunas armazena seu conteúdo de forma inversa aos bancos de dados orientados a linhas. Este formato de armazenar as informações torna-se vantajoso para Data Warehouses, onde agregações são processadas sobre uma quantidade de dados de características similares. Nesta categoria estão incluídos bancos, tais como Cassandra e Hbase (FindTheBest, 2012).

2.1.3 Orientado a documentos

Um banco de dados orientado a documentos armazena, recupera e gerencia dados semiestruturados. O elemento dos dados é chamado documento. Os documentos são endereçados no banco de dados via uma chave única que representa o documento. Uma das características de um banco de dados orientado a documentos é que, além da simples chave/valor de pesquisa, é possível recuperar um documento através de uma API, ou linguagem de consulta disponibilizada pelo banco. Todos os banco de dados correntes fornecem suporte a documentos no formato JSON (Leavitt, 2010).

O JSON é um padrão aberto de estruturação de dados baseado em texto e legível por humanos, utilizado para a serialização de dados. Foi projetado com o objetivo de ser simples, portátil, textual, e um subconjunto do *JavaScript* e tem se mostrado uma ótima alternativa ao XML, inclusive sendo nativo de armazenamento em alguns bancos de dados (JSON, 2012).

Um exemplo de documento JSON pode ser visto na Figura 2.1.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
  },
  "phoneNumber": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```

Figura 2.1: Exemplo documento JSON (Wikipedia, 2012)

Outro formato de documento utilizado é o BSON, semelhante ao JSON, porém com armazenado em binário, consistindo em uma lista de elementos chave/valor, onde a chave é uma string e o valor pode ser string, inteiro, data, booleano, nulo, objeto BSON ou matriz BSON (BSON, 2012).

Nessa categoria estão incluídos bancos, tais como: MongoDB, CouchDB,

Couchbase, RavenDB, SimpleDB, etc. (FindTheBest, 2012).

2.1.4 Baseado em Grafos

O armazenamento baseado em grafos fundamenta-se na teoria dos grafos. Em geral, vemos que grafo consiste de nós, propriedades e arestas. Os nós representam as entidades, as propriedades representam os atributos e as arestas representam as relações (Leavitt, 2010). Como exemplos de bancos de dados desta categoria podemos citar o InfoGrid e Neo4j (FindTheBest, 2012).

2.2 SGBDs NOSQL

Nesta seção encontra-se o estudo dos Bancos de Dados NoSQL pesquisados para a realização deste trabalho. Os bancos escolhidos para análise foram aqueles que frequentemente são citados como exemplos nas pesquisas anteriores.

2.2.1 Redis

O *REmote DIctionary Service* (Redis) é um banco de dados de código aberto do tipo chave/valor semi-persistente. Atualmente está na versão 2.6.2. Seu código é escrito em linguagem C sob a licença BSD e funciona na maioria dos sistemas POSIX, como linux, BSD, Mac OS e Solaris.

O Redis está presente em diversos serviços, entre eles o gitHub, o StackOverflow, além de outros. Seu melhor uso é em sistemas que exigem rápida troca de dados e muita frequência de escrita, como por exemplo, em aplicações com dados em tempo real, preço das ações, etc. (Seguin, 2012).

Esse banco de dados é referido, inclusive pelo seu autor, como um servidor de estrutura de dados, onde suas chaves podem conter strings, *hashes*, listas, conjuntos e conjuntos ordenados.

Sendo o Redis um banco de dados com armazenamento em memória, o seu ponto forte é a velocidade de leitura e gravação, podendo chegar a mais de 100.000 operações por segundo (Redmond, 2012).

2.2.1.1 Arquitetura

O Redis é um banco de dados do tipo cliente/servidor. Ele possui um servidor que recebe conexões, por padrão na porta 6379, e um cliente que se envia comandos através de um protocolo de comunicação. A Figura 2.2 mostra o diagrama da arquitetura.

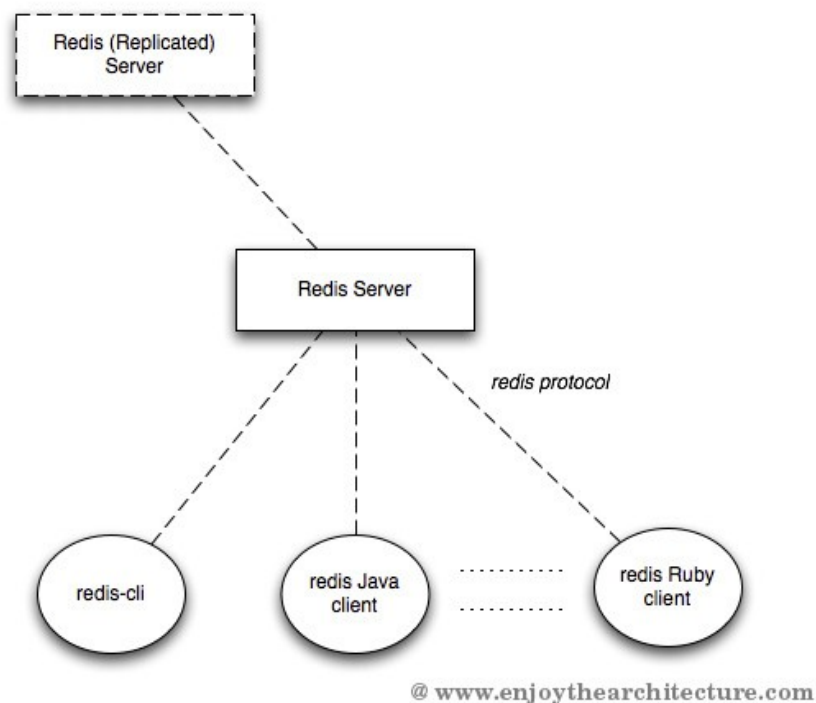


Figura 2.2: Arquitetura Redis (Campanelli, 2011).

O Redis possui um servidor executando um serviço, o **redis-server**, que “conversa” com os clientes através de um protocolo redis. Esse protocolo deve criar uma conexão TCP possibilitando que os clientes enviem comandos ao servidor.

O cliente oficial do redis é o **redis-cli**, desenvolvido em C e executado por linha de comando, mas têm sido desenvolvidos outros projetos em diferentes linguagens como C++, Haskell, Java, Objective-C, Perl, PHP, Ruby, Python, etc. No site oficial do Redis há uma relação dos projetos de clientes (Redis, 2010).

Os clientes possuem uma série de grupos de comandos que podem ser executados. Esses comandos são enviados ao servidor, que recebe, processa e envia ao cliente uma

resposta. Assim como outros bancos NoSQL, o Redis é completamente comprometido com velocidade, faz pouco uso de recursos, de segurança e opções de configurações triviais para ganhos de escalabilidade.

Nota-se também que é possível implementar uma arquitetura *master/slave* no Redis *Server* para ganho de desempenho, sendo possível replicação dos dados, armazenados em memória RAM, para outros servidores, caso faz-se necessário. Vale ressaltar que o importante é a quantidade de memória RAM disponível para o banco de dados, pois todos os dados são armazenados em memória, inclusive em uma replicação de servidores (Redmond, 2012).

Outra técnica que pode ser utilizada é a persistência dos dados da memória do servidor *master* em disco de um servidor *slave*, evitando uma latência maior no processamento do servidor principal.

2.2.1.2 Armazenamento

No que diz respeito ao armazenamento, pode-se dizer que o Redis é semipersistente, isso porque os dados são obrigatoriamente mantidos na memória RAM. Isso é um fator limitante para a quantidade de dados armazenados (Ippolito, 2009). Entretanto, também é possível persistir os dados em disco. Esse procedimento torna-se importante para uma possível recuperação, à medida que Redis não oferece tolerância a falhas e os dados mantidos em memória serão perdidos em caso de falha no servidor (Strauch, 2011).

O Redis é um banco de dados do tipo chave/valor, sendo possível trabalhar com *strings*, coleções de listas, de conjuntos, conjuntos ordenados e *hashes*. No entanto, tudo é representado e armazenado na memória como *string* (Tiwari, 2011). De maneira atômica, pode-se fazer operação de união, intersecção e diferenças entre conjuntos, além de trabalhar com listas, adicionando e removendo elementos de forma organizada.

Para o armazenamento em disco, há duas formas de persistir os dados. Uma delas é, de tempos em tempos, gravar uma cópia do conteúdo da memória RAM em disco. Esse armazenamento é conhecido como persistência RDB. A outra é a persistência AOF (*Append Only File*), onde cada comando enviado ao servidor é anexado em um arquivo de *log* (Redis, 2012).

Na persistência RDB são criados *snapshots* do conjunto de dados que estão na memória, que serão guardados em disco em um arquivo. Esse procedimento pode ser

realizado periodicamente caso um número mínimo de alterações tenham sido realizadas. Como alternativa, pode ser chamado manualmente através dos comandos **SAVE** ou **BGSAVE** do Redis. Já na persistência AOF todos os comandos recebidos pelo servidor são guardados em arquivo de *log*. Este arquivo pode ser usado para recuperação em caso de falha, reconstruindo o conjunto de dados original (Redmond, 2012).

Como foi comentado anteriormente, é possível utilizar vários servidores para armazenamento dos dados em memória. Também é possível utilizar servidores *slaves* para efetuar persistência dos dados, evitando a perda de desempenho com I/O no servidor principal.

2.2.1.3 *Download e instalação*

A seguir será descrito como efetuar o download do banco de dados Redis e o procedimento para sua execução. A instalação será efetuada no sistema operacional linux, utilizando a distribuição Ubuntu 10.04.

Em um terminal, digite os seguintes comandos:

```
$ wget http://redis.googlecode.com/files/redis-2.4.6.tar.gz
```

```
$ tar xzf redis-2.4.6.tar.gz
```

```
$ cd redis-2.4.6
```

```
$ make
```

Para iniciar o servidor:

```
$ src/redis-server
```

O cliente oficial pode ser inicializado da seguinte forma:

```
$ src/redis-cli
```

O Redis possui muitos comandos que podem ser executados, para criar listas, coleções, conjuntos. A seguir podemos testá-lo com o cliente:

```
redis> set nome João
```

```
OK
```

```
redis> get nome
```

```
"João"
```

2.2.2 Riak

Riak é um banco de dados do tipo chave/valor, distribuído e escalável. Ele foi desenvolvido pela empresa Basho Technologies, encontra-se atualmente na versão 1.2.0. Ele é escrito na linguagem Erlang e está sob a licença Apache 2.0. Ele possui pacote para instalação nos sistemas operacionais *Linux*, Mac OS e Solaris. (Riak, 2012)

2.2.2.1 Arquitetura

É um banco de dados totalmente distribuído. Ele pode ser acessado através do protocolo HTTP/REST ou de uma interface Erlang nativa. No caso de uma aplicação desenvolvida em Erlang, o acesso com a interface nativa pode ser simplificado, nos outros casos convém utilizar a API HTTP/REST, já que a maioria das linguagens tem embutidas primitivas para requisitar recursos sobre HTTP. Há também uma série de bibliotecas para clientes em várias linguagens de programação, como Java, Python e Ruby. Todos os clientes oferecem suporte para os métodos habituais de HTTP: PUT, GET, POST e DELETE.

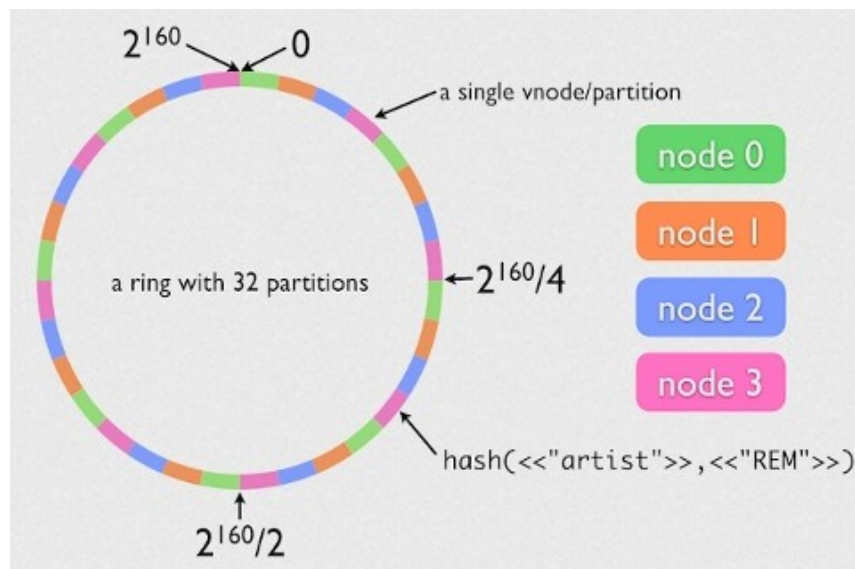


Figura 2.3: Modelo de Cluster no Riak (Riak, 2012).

O conceito de *cluster* é o ponto central no banco de dados Riak, sendo que não existe o conceito de nó *master*, ou seja, todos os nós são iguais, o que dá uma maior resistência a falhas, desde que exista a replicação dos dados. Cada *cluster* é um espaço inteiro de 160 bits

de *hashes* binários de pares *bucket/chave*, divididos em partições iguais. Os servidores físicos, referidos no *cluster* como nós, executam certo número de nós virtuais, os chamados “vnodes”. Cada “vnode” requisita uma partição deste anel e é responsável por armazenar uma porção separada do espaço das chaves. O número de “vnodes” por nó é calculado por número de partições/número de nós.

Na Figura 2.3 pode-se ter uma noção de como é o *cluster*, supondo um anel com 32 partições em 4 nós físicos, cada nó físico será responsável por 8 “vnodes”.

Os nós podem ser adicionados ou removidos do *cluster* de forma dinâmica, sem reiniciar o serviço, e o Riak se encarregará da distribuição dos dados. A replicação faz parte nativamente do Riak (Riak, 2012).

2.2.2.2 Armazenamento

Quanto ao armazenamento, Riak é composto por três conceitos: *buckets*, chaves e valores. *Buckets* são semelhantes à coleções, ou a tabelas, e agrupam objetos relacionados ou semelhantes. Os *buckets* possuem algumas propriedades que definem como os objetos são armazenados nele, como, por exemplo, a variável “*n_val*” que define quantas vezes um objeto será replicado no cluster, sendo útil para o caso de falha em um nó.

Buckets e chaves são as únicas maneiras de organizar os dados no Riak. Os dados são armazenados e referenciados pelo par *bucket/chave*. Cada chave está ligada a um único valor que pode ser de qualquer tipo, desde texto plano, arquivos binários, html ou até mesmo documentos no estilo JSON, inclusive, sendo referenciado por alguns como uma espécie de armazenamento do tipo orientado a documentos (Riak, 2012).

2.2.2.3 Download e instalação

Possui pacotes para instalação em sistema *linux*, Mac OS e Solaris. Para instalar o Riak, basta acessar o *site* <http://basho.com/resources/downloads/> e efetuar o download do pacote específico. No caso, a instalação foi realizada no Ubuntu 10.04.

Depois de efetuado o download, basta indicar o comando de instalação:

```
$ dpkg -i riak_1.2.0-1_i386.deb
```


Após isso pode-se inserir algum elemento de teste, onde a inserção é efetuada baseada no modelo: `http://SERVER:PORT/riak/BUCKET/KEY`

```
$ curl -v -X PUT http://localhost:8098/riak/favs/db \  
-H "Content-Type: text/html" \  
-d "<html><body><h1>My new favorite DB is RIAK</h1></body></html>"
```

Assim, foi armazenado no bucket “favs”, com chave “db” e o valor, que no caso foi o conteúdo de um documento html.

Na sequência vamos incluir um documento estilo JSON com o seguinte conteúdo: `{"nickname" : "The Wonder Dog", "breed" : "German Shepherd"}`

```
$ curl -v -X PUT http://localhost:8098/riak/animals/ace \  
-H "Content-Type: application/json" \  
-d '{"nickname" : "The Wonder Dog", "breed" : "German Shepherd"}'
```

No *site* do Riak <http://docs.basho.com/riak/latest/> é possível encontrar alguma documentação sobre a utilização do Riak.

2.2.3 Cassandra

Cassandra é um banco de dados de armazenamento de dados distribuído, orientado a colunas. Escrito em Java, foi desenvolvido inicialmente pelo Facebook e, atualmente, está na versão 1.1.5. É mantido por desenvolvedores da Apache e colaboradores. Está sob a licença Apache 2.0.

2.2.3.1 Arquitetura

O Banco de Dados Cassandra foi projetado para ser distribuído ao longo de várias máquinas que operam em conjunto, como uma única instância ao usuário final. De maneira simplificada, pode-se dizer que Cassandra é composto por um *cluster*, nós, *keyspaces* e família de colunas.

A estrutura mais externa é o *cluster*, também chamado de anel, como pode ser visto na Figura 2.4, isso porque Cassandra atribui dados aos nós do *cluster* organizando-os em forma de anel. Um *cluster* é um contêiner para *keyspaces*.

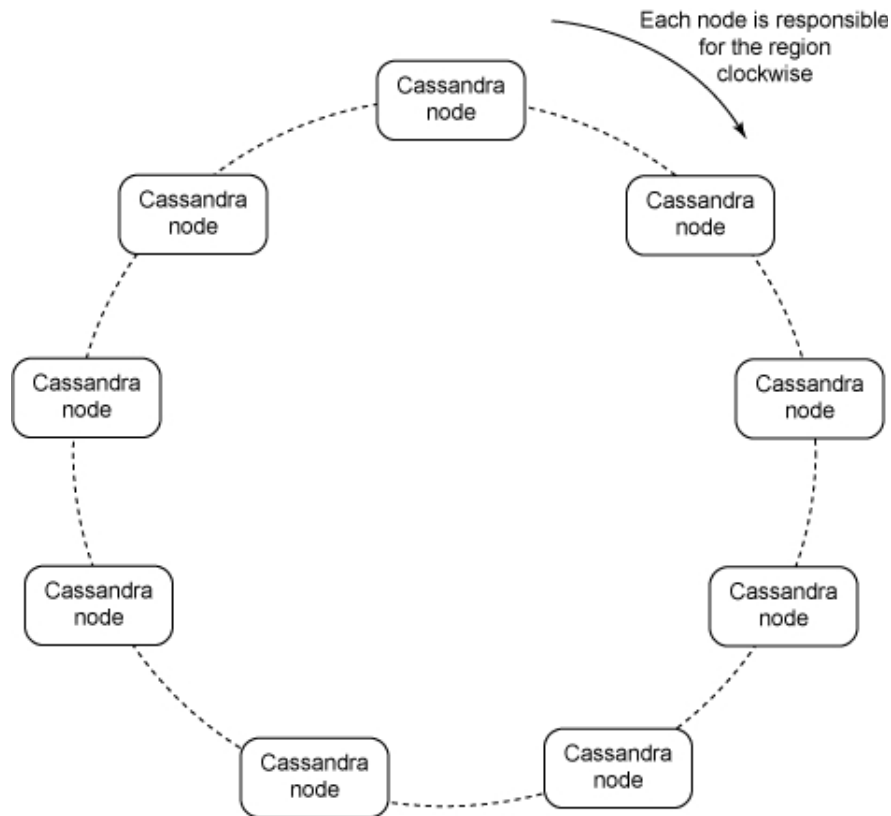


Figura 2.4: Cluster do Cassandra (Cassandra, 2012)

Os dados são distribuídos de forma transparente, qualquer nó aceita qualquer solicitação (leitura, gravação ou exclusão) e encaminha ao nó correto, sendo que não existe distinção, todos os nós são logicamente iguais. É possível ter um *cluster* com muitos nós, que é capaz de lidar com uma quantidade enorme de dados, na casa dos *petabytes*, por exemplo.

2.2.3.2 Armazenamento

O *Keyspace* é o recipiente para os dados do aplicativo, semelhante a um banco de dados no modelo relacional. Da mesma forma que um banco de dados é um contêiner para tabelas, um *keyspace* é um contêiner para uma lista de uma ou mais famílias de colunas.

Keyspace possui nome e conjunto de atributos que definem o seu comportamento. É possível configurar alguns atributos básicos como, por exemplo, o “*Replication Factor*” que define o número de nós que agirão como cópias de cada registro inserido. É importante ressaltar que a cada quanto maior este número, menor o desempenho. Outra observação faz-se quanto ao número de *keyspaces* por aplicação, nada impede que sejam criadas múltiplas *keyspaces*, entretanto, recomenda-se o uso de apenas uma por aplicação (HEWITT, 2010).

Se o *keyspace* é um contêiner para família de colunas, uma família de coluna é contêiner para uma coleção ordenada de linhas, cada uma das quais é uma coleção ordenada de colunas.

Apache Cassandra é considerado livre de esquema, ou seja, as colunas não necessariamente precisam conter os mesmos campos, sendo possível adicionar livremente qualquer coluna, de acordo com a necessidade, para qualquer família de coluna.

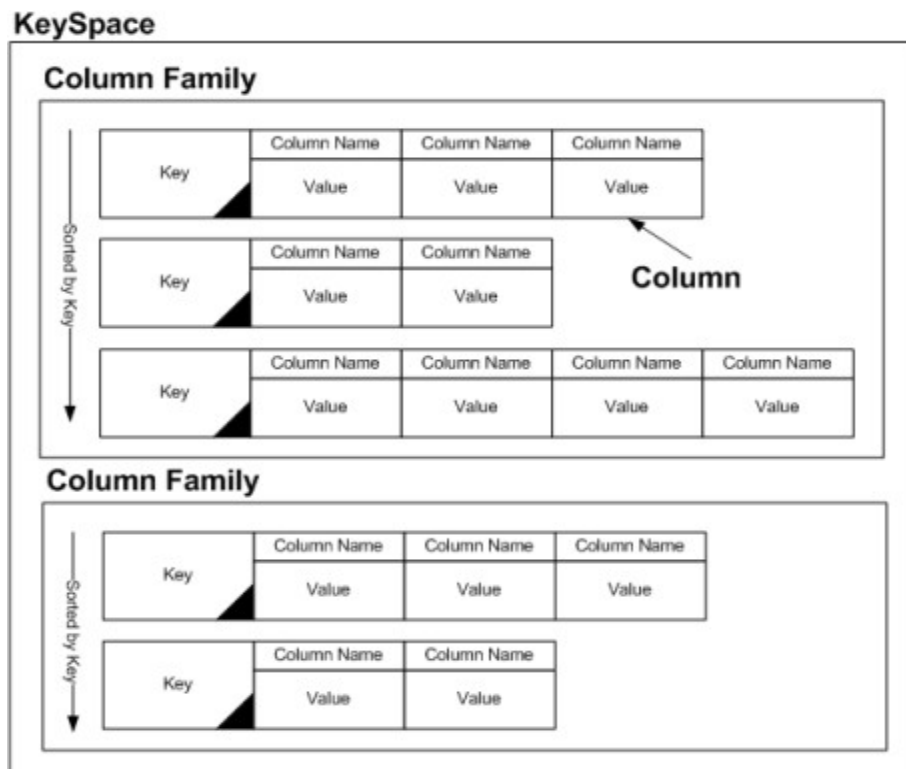


Figura 2.5: Modelo de dados do Apache Cassandra (Cho, 2010)

Na Figura 2.5 podem ser visualizados todos os objetos mencionados acima. Há um *keyspace*, duas famílias de colunas contendo linhas que não seguem um esquema fixo.

Os dados são escritos em estruturas de memória (*memtables*) e, em seguida, escritas em disco, uma vez que a estrutura de memória estiver cheia. *Memtable* possui tamanho padrão de 128MB. Há algumas questões a ressaltar, todos os dados de uma linha devem

caber em uma única máquina do *cluster*, para que esse seja replicado. O tamanho máximo de uma coluna não pode ultrapassar 2GB e o podem existir no máximo 2 bilhões de colunas por linha.

Como mencionado anteriormente, Apache Cassandra é um ótimo servidor para grandes quantidades de dados, com fácil processo de gravação.

2.2.3.3 Download e instalação

Antes de instalar o Cassandra, é preciso verificar a existência do Java instalado na máquina servidora.

Para a instalação, recomenda-se adicionar o endereço dos repositórios ao arquivo de configuração do Ubuntu “/etc/apt/sources.list”, como mostra a Figura 2.6.

```
deb http://www.apache.org/dist/cassandra/debian 11x main
deb-src http://www.apache.org/dist/cassandra/debian 11x main
```

Figura 2.6: Repositório do Apache Cassandra

Após isso, pode-se fazer o *download* da chave pública do repositório, seguindo o exemplo da Figura 2.7:

```
gpg --keyserver pgp.mit.edu --recv-keys 2B5C1B00
gpg --export --armor 2B5C1B00 | sudo apt-key add -
```

Figura 2.7: Chave Pública do repositório

Na Figura 2.8, podem ser vistos os comandos para a instalação do Cassandra no Ubuntu 10.04:

```
sudo apt-get update
sudo apt-get install cassandra
```

Figura 2.8: Instalação do Apache Cassandra

O Cassandra possui um cliente padrão para console, que pode ser iniciado pelo terminal através do comando:

```
$ cassandra-cli -host localhost -port 9160
```

Para testar, pode-se fazê-lo informando alguns comandos básicos de inserção, busca, etc. Para isto, cria-se um novo *keyspace* chamado “teste”, por exemplo:

```
$ create keyspace teste;
```

A seguir, criar uma nova família de colunas, com o nome “User”:

```
$ create column family Usuario with comparator = UTF8Type;  
    AND column_metadata = [  
        { column_name: primeiro, validation_class: UTF8Type},  
        { column_name: ultimo, validation_class: UTF8Type},  
        { column_name: idade, validation_class: UTF8Type, index_type: KEYS}  
    ];
```

Neste exemplo, as colunas são estáticas, porque foram definidas anteriormente. Para adicionar um elemento, pode-se fazer da seguinte forma:

```
$ set Usuario['jsilva']['primeiro'] = 'Joao';  
$ set Usuario['jsilva']['ultimo'] = 'Silva';  
$ set Usuario['jsilva']['idade'] = '56';
```

A seguir um exemplo de como criar uma família de colunas dinâmicas:

```
$ create column family Usuario  
    WITH comparator = TimeUUIDType  
    AND key_validation_class=UTF8Type  
    AND default_validation_class = UTF8Type;
```

Para consultar algum registro, retornando, por exemplo, o registro anteriormente inserido, pode-se realizar da seguinte forma:

```
$ get Usuario where idade = '56';
```

Existe também uma ferramenta gráfica para administrar os *clusters* no Apache Cassandra, semelhante ao phpMyAdmin, o Cassandra Cluster Admin. Com ele é possível criar, editar, excluir *keyspaces*, famílias de colunas, exibir, procurar dados e inserir linhas. Entretanto, a ferramenta não foi testada.

Além do cliente padrão, existem clientes para Java, PHP, C++, Perl, entre outros. Há também suporte à linguagem de consulta CQL, semelhante à SQL, tendo *drivers* disponíveis para Java e Python.

2.2.4 HBase

Hbase é um banco de dados orientado a colunas, altamente escalável e distribuído, que suporta armazenamento de dados estruturados para grandes tabelas. Foi desenvolvido, e é mantido pela Apache Software Foundation (ASF). É escrito na linguagem Java, sob a licença Apache 2.0. Está atualmente na versão 0.94. Sua utilização é recomendável quando é necessário acesso de leitura/escrita em tempo real e aleatório para grandes dados (HBase, 2012).

2.2.4.1 Arquitetura

O Hbase é um subprojeto do Apache Hadoop, que desenvolve softwares *open source* para computação distribuída e escalável. De outro subprojeto do Apache Hadoop vêm o HDFS, sistema de arquivos que oferece acesso com alta taxa de transferência para os dados da aplicação. O Hbase pode ser executado tanto em sistemas de arquivos comuns, como no sistema de arquivos HDFS, que também é desenvolvido pela Apache e oferece acesso com alta taxa de transferência para os dados da aplicação. Entretanto, com o HDFS é possível ter uma gama maior de recursos para o Hbase, isto porque são sistemas desenvolvidos em paralelo e pela mesma empresa.

Existem 3 modos de executar o servidor Hbase:

- *Stand-alone*, onde uma máquina trabalha sozinha;
- *Pseudodistributed*, onde um nó “fingindo” ser um cluster;
- *Fully distributed* é o modo onde há vários nós trabalhando em conjunto.

Na Figura 2.9 é possível visualizar a arquitetura do HBase.

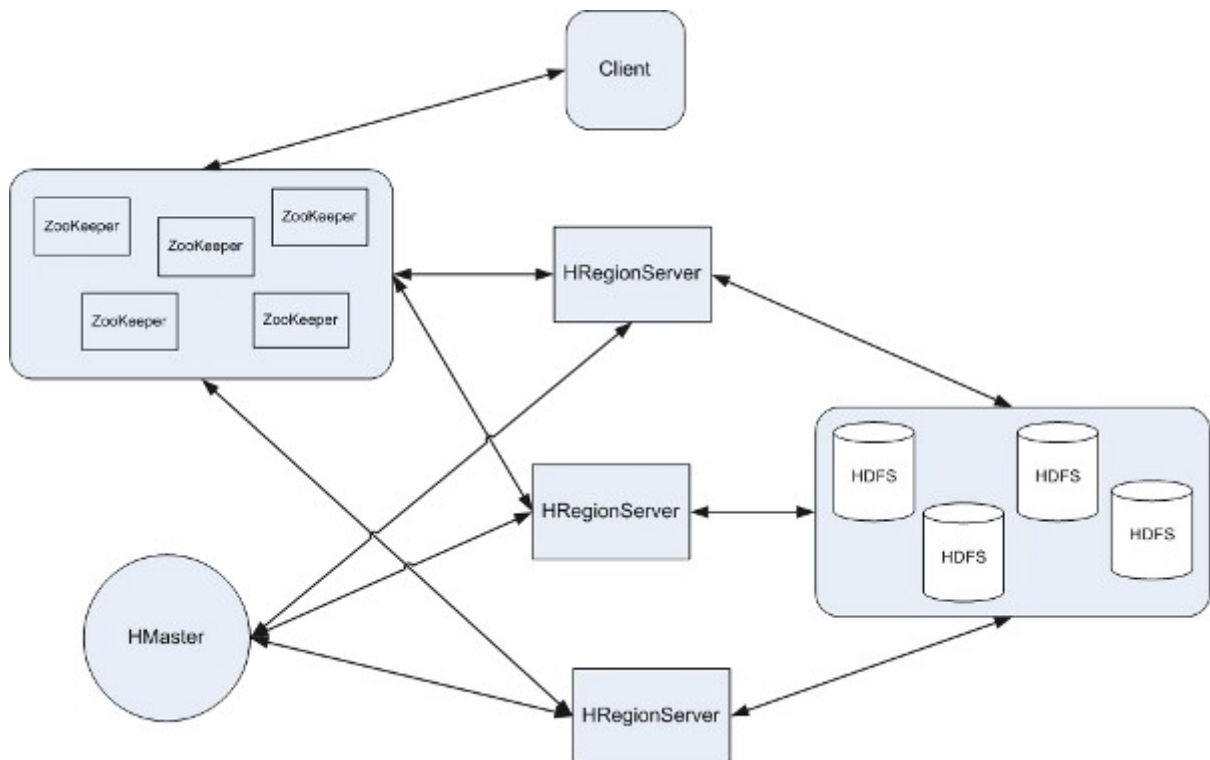


Figura 2.9: Arquitetura HBase.

Para executar a forma totalmente distribuída é preciso ter um cluster ZooKeeper rodando. ZooKeeper é um serviço de Coordenação para aplicações distribuídas. Ainda é necessário ter um servidor master e vários servidores regionais como mostrado na Figura 2.X

2.2.4.2 Armazenamento

O HBase suporta dados não estruturados e parcialmente estruturados. Para fazer isso, os dados são organizados em famílias de colunas. É endereçado um registro individual, chamado de “célula” no Hbase, com uma combinação de *row*, *column* e *timestamp*. Não é necessário definir a tabela com antecedência, pode-se simplesmente nomear uma família de colunas e então permitir que a célula se qualifique para ser determinada em tempo de execução. Isso permite que você seja bastante flexível e suporta uma abordagem ágil de desenvolvimento.

2.2.4.3 Download e instalação

O Hbase está disponível oficialmente para Linux e o *download* pode ser realizado no endereço <http://www.apache.org/dyn/closer.cgi/hbase/>. Entretanto, não foi obtido sucesso na instalação do *software* no Ubuntu 10.04, mesmo que tenha suporte para este sistema operacional. Eram exigidas muitas configurações adicionais, importações de bibliotecas e instalação do Java. Mesmo realizando o procedimento indicado na documentação, não se obteve êxito.

2.2.5 CouchDB

Apache CouchDB é um banco de dados escalável, tolerante a falhas e orientado a documentos de esquema livre. Foi desenvolvido por Damien Kat, ex-desenvolvedor da IBM, em 2005. Posteriormente, em 2008, foi repassado à Apache, que deu continuidade ao projeto. É escrito na linguagem Erlang e está sob a licença Apache 2.0. Está na versão 1.2 e possui pacote de instalação para Linux, Mac Os e, recentemente, também para Windows (CouchDB, 2012).

2.2.5.1 Arquitetura

O CouchDB utiliza uma API HTTP/REST para a manipulação dos dados, portanto a comunicação entre cliente e servidor pode ser realizada em qualquer aplicação que possa comunicar-se por HTTP. As operações básicas de um banco de dados são mapeadas para as operações do protocolo HTTP, como GET, PUT, POST, DELETE, para ler, armazenar, substituir e remover objetos armazenados.

Além da API REST/HTTP que provê acesso a qualquer aplicação com suporte ao protocolo HTTP, o CouchDB oferece um servidor web, chamado Futon, que pode ser acessado pelo *browser* através do endereço http://localhost:5984/_utils/, onde é possível gerenciar os bancos e documentos individualmente.

Na Figura 2.X tem-se um modelo de arquitetura do CouchDB, indicando que a requisição que é feita através do cliente HTTP, chegando ao módulo do CouchDB que trata das requisições. Tem-se também os processos que estão rodando no servidor e são

responsáveis pelas tarefas de replicação, visualização e armazenamento dos dados.

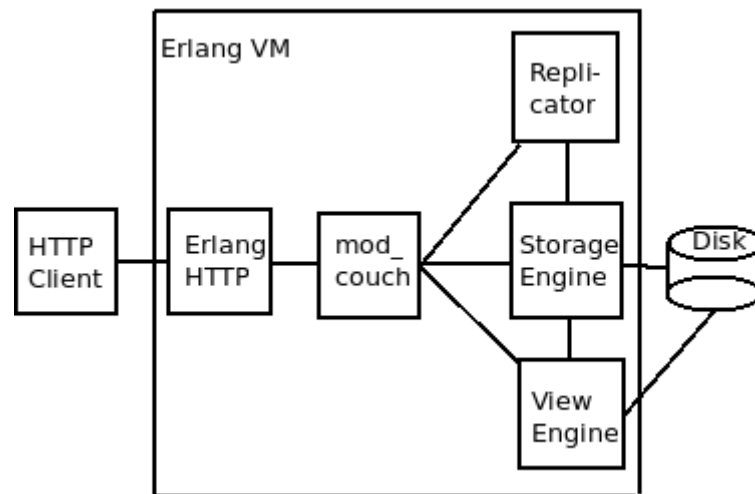


Figura 2.10: Arquitetura CouchDB.

A arquitetura do CouchDB consiste em um servidor de banco de dados *master* que possui um ou mais bancos de dados criados. Os bancos são compostos por objetos, que são documentos do tipo JSON, de esquema livre, ou seja, em uma mesma base de dados os documentos não precisam ter os mesmos campos. Entretanto, recomenda-se armazenar documentos semelhantes em um mesmo banco de dados.

CouchDB suporta múltiplas réplicas do banco de dados rodando em diferentes servidores e fornece um mecanismo para sincronização destes dados. Há um processo, replicador, executando que possibilita informar a origem e o destino do servidor, é realizada uma verificação pela data e caso houve mudanças, a cópia é sobrescrita no servidor *slave*. As replicações são realizadas de forma assíncrona, de forma que não compromete a velocidade de escrita e leitura (ANDERSON, 2010).

2.2.5.2 Armazenamento

Os documentos são a unidade primária de dados no CouchDB e são constituídos de campos de pares chave ou nome e valor. Os nomes das chaves devem ser únicos dentro de um documento e o valor atribuído pode ser uma *string*, número, *booleano*, data, listas, entretanto, não podem existir documentos aninhados. Existem dois campos fixos em qualquer documento, que servem para identificação única, um ID do documento e um número de revisão, onde o ID é único para qualquer novo documento e o campo revisor

serve para identificar as várias modificações realizadas em um documento (Strauch, 2011).

Um ponto importante sobre o armazenamento é que a edição de um documento na verdade é a criação uma nova versão do documento, com o mesmo identificador ID, com as alterações efetuadas e um novo número de revisão. Isto pode ser útil muitas vezes, como para recuperação de dados em caso de falha, mas também é desperdiçado muito espaço em disco com arquivos antigos e até certo ponto inúteis. As modificações só podem ser realizadas na última versão do documento. Também, não é possível excluir uma versão, somente o documento por completo.

Para atualizações de documentos é utilizado o modelo *lockless*, onde o documento é carregado na aplicação cliente onde se realiza a edição, e guarda-se no banco de dados novamente. Caso outro cliente editou o mesmo documento, será recebido uma mensagem de conflito de edição ao tentar salvar. Para resolver este problema, a versão mais recente do documento poderá ser aberta, isso acontece porque não é oferecido mecanismo de bloqueio na escrita. As atualizações de documentos (inserção, edição ou remoção) são atômicas, ou seja, são totalmente salvos ou nada é salvo, não existem salvamentos parciais.

Os documentos são armazenados utilizando um mecanismo de indexação sobre estruturas de Árvores B, que são excelentes estruturas para manter os dados classificados e propiciam rapidez nas buscas, inserções e exclusões de documentos (ANDERSON, 2010).

2.2.5.3 *Download e instalação*

É possível fazer o *download* através do *site* oficial <http://couchdb.apache.org/>. Há pacotes disponíveis para Linux, e Mac OS e, desde 2010, também para Windows. No Ubuntu 10.04, já está incluído na central de *softwares*, então pode ser instalado pelo comando “`aptitude install couchdb`”.

Como o CouchDB tem uma API REST, a conexão entre cliente e servidor pode ser realizada entre qualquer aplicação que possua suporte ao protocolo HTTP. Nos exemplos, será utilizado a ferramenta *cURL*.

Criando um novo banco de dados “Usuario”:

```
$ curl -X PUT http://127.0.0.1:5984/usuario
```

Inserindo um documento banco de dados:

```
$ curl -X PUT http://127.0.0.1:5984/usuario/6e1295ed6c29495e54cc05947f18c8af  
\ -d '{"Nome": "Joao da Silva", "Idade": "56"}'
```

2.2.6 MongoDB

O MongoDB é um banco de dados orientado a documentos, livre de esquema, escrito em C++, sob a licença AGPL versão 3.0. Foi desenvolvido em um projeto *open source*, impulsionado principalmente pela empresa 10gen, que também oferece serviços profissionais em torno MongoDB. Atualmente, continua sendo mantido pela mesma empresa, está na versão 2.2.2 e em constante desenvolvimento. O seu nome, Mongo, deriva do adjetivo *humongous*, monstruoso em português (MongoDB, 2012).

De acordo com seus desenvolvedores, o objetivo principal do MongoDB é preencher a lacuna entre os bancos de dados de armazenamento chave/valor, rápidos e altamente escaláveis, e os tradicionais sistemas de gerenciamento de banco de dados relacionais, que possuem grande quantidade de recursos. Assim, MongoDB foi projetado para otimização de desempenho, facilidade de escalabilidade horizontal, alta disponibilidade e capacidade de consultas avançadas (Strauch, 2011).

2.2.6.1 Arquitetura

O aplicativo MongoDB é composto por dois tipos de serviços, o processo servidor **mongod** que é o núcleo do banco de dados e o serviço **mongos** para *autosharding*. *Sharding* é a divisão dos dados em vários nós, sendo utilizado quando faz-se necessário balanceamento de carga. O processo servidor pode rodar tanto em arquiteturas 32 como 64-bit, no entanto, é recomendado o uso de 64-bit, uma vez que o tamanho máximo de um banco de dados é limitado à 2GB no modo 32-bit. Além dos serviços do aplicativo servidor, há também serviço cliente **mongo**, que é um cliente *shell* padrão do MongoDB utilizado para conectar-se ao servidor através da porta 27017. Entretanto, há uma grande variedade de *drivers* oficiais disponíveis, como C, C++, Haskell, Java, PHP, entre outros, todos eles estão sob a licença Apache. Também é possível conectar através da interface HTTP/REST na porta 28017, permitindo a manipulação de entradas via HTTP.

O servidor MongoDB pode hospedar mais de um banco de dados, independentes entre si, armazenados separadamente. Um banco de dados contém um ou mais coleções constituídas por documentos BSON (Binary JSON), que são estruturados como documentos

JSON, com esquema dinâmico, fazendo com que a integração de dados em certos tipos de aplicações sejam mais fáceis e rápidos (MongoDB, 2012).

As consultas são expressas em sintaxe como JSON e são enviadas ao servidor como objetos BSON pelo *driver* do banco de dados. O modelo permite consultas a todos os documentos dentro de uma coleção, incluindo objetos e matrizes incorporadas. MongoDB não possui transações, tampouco *joins*, ficando a cargo do desenvolvedor implementá-las, se necessário, em uma aplicação. É possível inclusive relacionar documentos de coleções distintas.

Quanto aos recursos disponíveis, vale ressaltar o suporte automático para a realização de **Sharding**. O sistema de banco de dados pode ser distribuído através de um *cluster* de máquinas. O *cluster* consiste em três componentes, nós *Shard*, servidores de configuração e serviços de roteamento chamados **mongos**. Na Figura 2.9, mostrada a seguir, é possível visualizar o *cluster*.

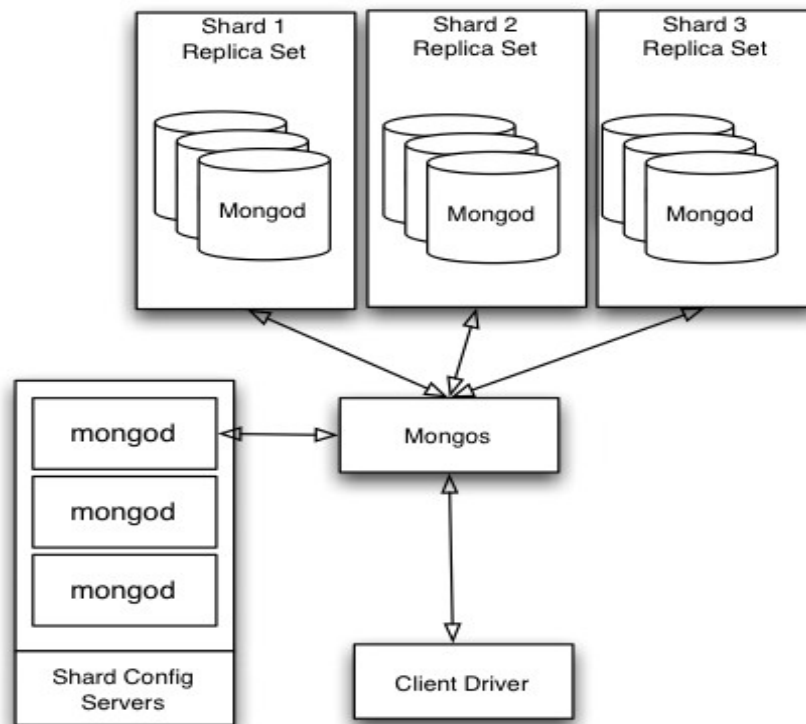


Figura 2.11: Arquitetura Cluster MongoDB (Orend, 2010).

Os nós *Shard* são responsáveis por armazenar os dados atuais, podendo inclusive ser replicados para redundância em caso de falha. Os servidores de configuração são usados para armazenar os metadados e rotear informação do *cluster* de MongoDB. Os *Mongos* são

responsáveis por rotear os processos, otimizando o desempenho das tarefas requisitadas pelos clientes (Orend, 2010).

Outro recurso importante é o **GridFS**, que serve para armazenar e recuperar arquivos que excedam o limite de 16M dos documentos BSON. Os arquivos são divididos em partes e armazenados em uma coleção, em outra coleção são armazenados os metadados do arquivo. É útil para armazenar arquivos grandes, como áudio e vídeo (MongoDB, 2012).

2.2.6.2 Armazenamento

Quanto ao armazenamento, de forma simplificada pode-se dizer que o servidor MongoDB contém um ou mais bancos de dados, independentes, armazenados separadamente. Cada banco de dados é composto por uma ou mais coleções constituídas de documentos estruturados BSON, estes, são armazenados como BSON objetos, que são binários codificados como objetos JSON. BSON suporta estrutura de objetos aninhados com objetos e matrizes embutidos, assim como JSON. Cabe ressaltar que o tamanho máximo de cada documento é 16MB (Strauch, 2011).

Como já mencionado, os documentos no MongoDB são organizados em coleções. Cada coleção pode conter qualquer tipo de documento, entretanto aconselha-se utilizar uma coleção para cada tipo de documento. Cada documento tem um campo de ID, que é usado como uma chave primária **ObjectID**. Estes índices são armazenados em estruturas como árvore B. Para permitir consultas rápidas, o desenvolvedor pode criar um índice para cada campo de consulta possível um documento. MongoDB também suporta indexação sobre os objetos embutidos e matrizes.

MongoDB usa arquivos de memória mapeada, que mapeia diretamente um arquivo armazenado em disco para um *array* de bytes em memória virtual, não memória física RAM, onde a lógica de acesso aos dados é implementado usando aritmética de ponteiro. O uso de arquivos mapeados em memória é eficiente para melhorar o desempenho (Orend, 2010).

2.2.6.3 Download e instalação

Estão disponíveis distribuições binárias para Windows, Mac OS X, Linux e Solaris.

Todo o processo de instalação e execução descrito aqui foi realizado no Sistema Operacional *Linux*, distro Ubuntu 10.04. Para outros sistemas operacionais, basta acessar o *site* <http://www.mongodb.org/downloads> e efetuar o download para a distribuição específica.

Para a instalação do MongoDB deve-se adicionar o endereço “**deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen**” ao arquivo de configuração onde estão listados os repositórios utilizados no Ubuntu, este arquivo é usualmente encontrado no diretório “/etc/apt/sources.list”. Em um terminal digite o comando “**aptitude install mongodb-10gen**”, isto instalará os pacotes necessários ao funcionamento do banco de dados.

No arquivo “/etc/mongo.conf” encontra-se as configurações do servidor, que podem ser alteradas de acordo com a necessidade, como porta, local de armazenamento dos dados, etc. Por padrão, os arquivos de dados são armazenados no diretório “/var/lib/mongodb” e o *log* em “/var/log/mongodb”.

Para iniciar o servidor basta chamar o processo mongod através do terminal:

```
$ service mongodb start
```

O serviço cliente pode ser inicializado através do comando:

```
$ mongo
```

Por padrão, o shell do Mongo se conecta ao banco de dados “test”, utilizando a interface localhost como padrão. Para conectar-se a diferentes bancos, servidor, etc. é necessário especificar explicitamente. Com o cliente já conectado ao banco, pode-se efetuar alguns testes, como inserção, remoção, busca, etc.

Primeiramente vamos alterar para outro banco de dados, para isto use o comando:

```
> use testeDB;
```

Com isto passaremos a trabalhar com o banco de dados “testeDB” que, caso ainda não exista, será criado no momento de inserção do primeiro documento. Outro detalhe é que não é necessário que a coleção exista no momento em que vamos inserir o documento, pois será criada quando da inserção do primeiro documento. Agora podemos inserir um documento para teste, o faremos no banco que “testeDB” que estamos trabalhando e na coleção, ainda inexistente, “testeColecao”. Para isto, executamos o comando:

```
> db.testeColecao.save({nome: "Joao da Silva", idade: "56"});
```

As consultas no MongoDB são baseadas em documentos, então podemos listar todos os documentos na coleção “testeColecao” com o seguinte comando:

```
> db.testeColecao.find( );
```

Para listar somente os documentos que contenham a chave idade, com valor 56 procedemos da seguinte maneira:

```
> db.testeColecao.find( {"idade": "56"});
```

O MongoDB possui ainda uma série de operadores para consultas avançadas como por exemplo (o significado entre parênteses): **\$lt** (menor que), **\$lte** (menor ou igual que) **\$gt** (maior que), **\$gte** (maior ou igual que), **\$ne** (diferente de), entre outros. Na documentação oficial é possível consultá-los.

2.2.7 Neo4j

Neo4j é um banco de dados de código aberto, orientado a grafos. Foi desenvolvido pela empresa Neo Technology, em 2007. É escrito na linguagem Java e possui dois tipos de licenças, uma de código aberto, GPLv3 e outra, com os módulos adicionais, comercial sob a licença AGPLv3.

É descrito por seus desenvolvedores como um banco de dados integrado, baseado em disco, com mecanismo de persistência em Java totalmente transacional que armazena dados estruturados em grafos. Apresenta transações ACID, alta disponibilidade (Neo4j, 2012).

2.2.7.1 Arquitetura e armazenamento

O Neo4j faz valer a conceito de orientação a grafos, permitindo que os dados sejam persistidos, percorridos e visualizados da mesma maneira que os grafos.

Ao contrário dos outros bancos de dados NoSQL, o Neo4j mantém algumas características que se tornaram comuns em bancos de dados relacionais, como controle de transações e a propriedade ACID.

As unidades fundamentais que formam um grafo são os nós e relacionamentos. No Neo4j, tanto nós como relacionamentos podem conter propriedades. As Relações entre nós são uma parte chave do banco de dados baseado em grafo, eles permitem encontrar dados relacionados.

Assim como os nós, relacionamentos também podem ter propriedades e um relacionamento que conecta dois nós, garante que o nó inicial e final são válidos. É interessante ressaltar que no Neo4j, os relacionamentos são sem esquema definido, ou seja,

podem ser de qualquer tipo.

As propriedades dos nós e relacionamentos são pares chave/valor, onde a chave é uma *string* e os valores podem ser tipos primitivos como *string*, *int*, *booleano*, ou matriz de um tipo primitivo. O caminho é um ou mais nós de de ligação com os relacionamentos (Redmond, 2012).

O banco de dados Neo4j é recomendável para situações em que seriam necessárias muitas operações, como pesquisas extremamente complexas nos bancos de dados relacionais. Neo4j baseia-se na teoria dos grafos.

Quanto a disponibilidade, Neo4j foi projetado para ser facilmente replicado entre servidor *master* e *slave* com uma simples mudança de configuração do serviço “**EmbeddedGraphDatabase**” para o modo de alta disponibilidade “**HighlyAvailableGraphDatabase**”. Ao executar neste modo, há sempre um único master e zero ou mais slaves. No entanto, o serviço de alta disponibilidade só está presente na edição comercial.

O Neo4j possui uma linguagem própria para consulta, a CQL, que permite uma expressiva e eficiente consulta a armazenamento em grafo. A linguagem é baseada em *pattern matching* e tem sintaxe semelhante ao SQL (Neo4j, 2012).

Na Figura 2.10 tem-se um exemplo de um modelo de banco de dados orientado a grafo.

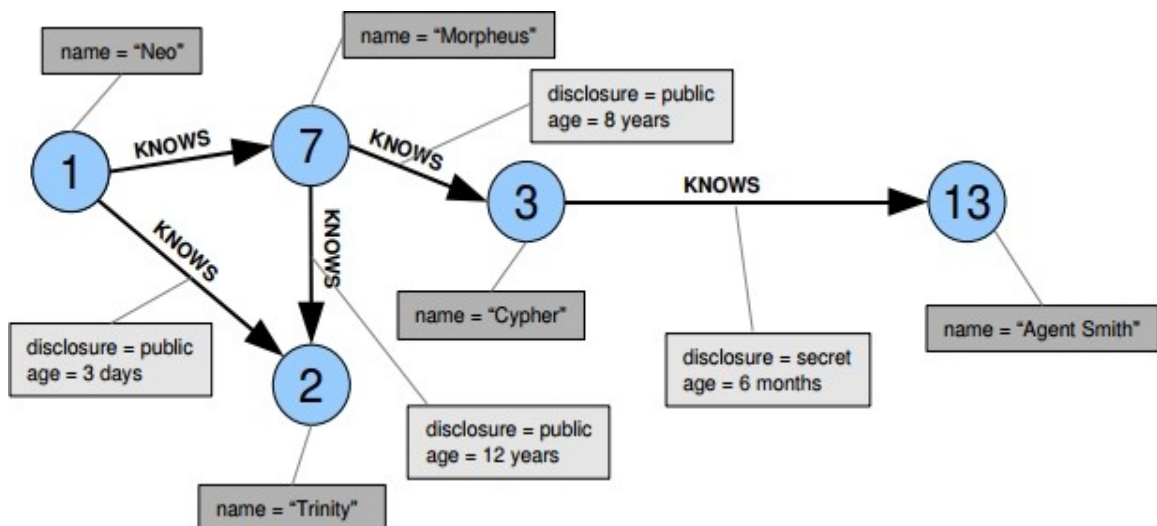


Figura 2.12: Modelo de dados no Neo4j (Neo4j, 2012).

No exemplo da Figura 2.10, existem vários nós (números 1, 2, 3, 7 e 13) e relacionamentos dirigidos declarando que uma pessoa conhece a outra, se esse

conhecimento é público ou secreto e o tempo em que se conhecem. Os nós possuem como propriedade um nome, as relações têm propriedade que descrevem o tempo que as pessoas se conhecem e dirigido declarando que uma pessoa sabe da outra e se o conhecimento é público ou não.

2.2.7.2 Download e instalação

Neo4j pode ser baixado diretamente do *site* oficial <http://www.neo4j.org/install>. Há versões para Linux, Mac e Windows.

Depois de baixar, basta extrair e executar. É necessário ter o java instalado.

```
root@note-ivan# tar -xzf neo4j.tar.gz
```

```
root@note-ivan# cd neo4j
```

```
root@note-ivan:/neo4j# ./bin/neo4j start
```

Apesar do Neo4j requerer java para ser executado, o acesso pode ser feito de qualquer aplicação que permita a comunicação HTTP, isso porque vem com uma interface REST. O acesso pode ser feito pelo navegador através do endereço <http://localhost:7474/webadmin>. A partir da interface gráfica pode ser realizado todo gerenciamento do banco de dados, como inclusão e remoção de nós e relacionamentos.

2.3 COMPARATIVO ENTRE OS BANCOS DE DADOS NOSQL

Após o estudo dos bancos de dados NoSQL, é apresentado nesta seção uma tabela comparativa sobre alguns itens. A comparação leva em consideração para quais sistemas operacionais têm-se pacotes de instalação disponíveis, o tipo de licença, o design da arquitetura, o modelo de dados utilizado, como os dados são armazenados, como é possível escalar ou replicar e como pode ser executado. Na Tabela 2.1 pode-se visualizar o quadro comparativo entre os bancos de dados NoSQL.

	Ano	Desenvol-vedor	Sistema Operacional	Licença	Arquitetura	Modelo de dados	Armazena-mento	Execução
Redis	2009	Salvatore Sanfilippo	Linux, Mac OS e Solaris	BSD	Não distribuída. É um servidor de estrutura de dados. Dados podem ser replicados em <i>master / slave</i>	Chave / valor	Em memória RAM. <i>Snapshots</i> em disco para durabilidade	Gerenciamento através de programas clientes em C, C++, Java, entre outros.
Riak	2010	Basho Technologies	Linux, Mac OS e Solaris	Apache 2.0	Distribuída. Nós físicos compostos de nós virtuais, os “vnodes”. Dados distribuídos automaticamente no <i>cluster</i> . Fácil adição de máquinas	Chave / valor	Em disco. <i>Buckets</i> com pares chave / valor.	API HTTP / REST. Cliente Erlang nativo ou clientes em diversas linguagens de programação, como java, php e C.
Cassandra	2008	Desenvolvido pelo Facebook. Mantido por ASF	Linux, Windows	Apache 2.0	Arquitetura distribuída. Possível adicionar nós ao <i>cluster</i> . Replicação de dados entre os nós.	Orientado a colunas	Configurável armazenamento em disco ou memória	Apache Thrift para acesso em muitas linguagens.
HBase	2009	ASF	Linux	Apache 2.0	Roda sobre o HDFS. Suporta tanto arquitetura totalmente distribuída, como independente	Orientado a colunas	Configurável armazenamento em disco ou memória	
CouchDB	2005	Damien Katz (ex IBM). Mantido por ASF	Linux, Mac OS e Windows	Apache 2.0	Arquitetura distribuída com replicação bidirecional (sincronização) dos dados.	Orientado a documentos	Documentos JSON em disco, árvores B para indexar. Mantém versões.	Oferece API REST/HTTP. Também um gerenciador web chamado Futon
MongoDB	2007	10gen	Windows, Mac OS X, Linux e Solaris	AGPL	Cliente / servidor. Serviço “mongos” define para qual servidor enviar requisição. Dados distribuídos automaticamente entre os nós.	Orientado a documentos	Coleção de documentos armazenados no disco, organizados em forma Árvore B	<i>Drivers</i> oficiais disponíveis para C, C++, Haskell, Java™, JavaScript, Perl, PHP, entre outros
Neo4j	2007	Neo Technology	Linux, Mac e Windows	GPLv, AGPL e comercial	Replicação dos dados <i>master / slave</i>	Orientado a grafos		API REST e gerenciamento <i>web</i>

Tabela 2.1: Tabela comparativa NoSQL Databases.

Pelos estudos realizados e pela Tabela 2.1 de comparação, é possível identificar que a maioria dos bancos de dados NoSQL testados são suportados pelos principais sistemas operacionais. A maioria deles também possui uma arquitetura distribuída, ou seja, estão presentes em vários nós no mesmo servidor, ou em servidores distintos.

Em relação ao armazenamento, alguns são simples pares de chave/valor, enquanto outros possuem uma estrutura mais complexa, como é o caso do Neo4j. À medida que as estruturas tornam-se mais complexas pode-se acabar tendo perda de desempenho na escrita dos dados.

Quanto ao gerenciamento, alguns possuem gerenciamento *web* nativo, enquanto a maioria pode ser acessada por clientes padrões no *shell* ou através de bibliotecas para as diversas linguagens de programação.

No capítulo seguinte, será descrito a escolha de um dos bancos de dados estudados, os motivos que levaram a esta escolha e a proposta de desenvolvimento de uma aplicação, que é outro objetivo deste trabalho.

3 PROPOSTA DO TRABALHO

Neste capítulo é apresentada a proposta para o desenvolvimento de uma aplicação que utilizará um dos Bancos de Dados NoSQL estudados. A aplicação serve para consultar, inserir, alterar e remover objetos da base de dados, demonstrando o funcionamento em um sistema real.

3.1 PROPOSTA DE DESENVOLVIMENTO

A aplicação desenvolvida tem com objetivo o acesso a um banco de dados NoSQL, efetuando consultas, inserções, edições e remoções de registros da base de dados.

Para o desenvolvimento da aplicação foi escolhido o Banco de Dados MongoDB. A escolha deu-se pelo fato do MongoDB ser um banco de dados relativamente fácil de ser instalado e de simples entendimento, ideal para quando se está iniciando em projetos com Banco de Dados NoSQL. Inclusive pode ser recomendado para quem está fazendo uma transição de bancos de dados relacionais para NoSQL.

Outros pontos levados em consideração é que o banco de dados escolhido deveria ser gratuito, preferencialmente sob uma licença *open source*, estar disponível para Sistema Operacional *Linux* e possuir pacotes para os outros sistemas operacionais seria um diferencial. Além disso, MongoDB trabalha com documento BSON, que são documentos como JSON, porém guardados em forma binária. Estes documentos são um formato leve e rápido de intercâmbio de dados, sendo inclusive tratados como sucessor do XML. E, por fim, a documentação existente no *site* oficial também é bastante completa, o que é excelente

para começar o estudo de uma nova tecnologia e, há alguns livros sobre o assunto.

A aplicação foi desenvolvida na linguagem de programação Java, com interface gráfica. Para o desenvolvimento do código e a interface gráfica foi utilizado o software NetBeans IDE 7.1 e a interface gráfica utiliza a ferramenta *Swing*. Além disso, é utilizado o *driver* para Java para comunicação entre cliente e servidor. Tanto o *driver* como a documentação encontram-se disponíveis no *site* oficial do MongoDB. O *driver* está sob a licença Apache.

A aplicação consiste em um Repositório de Monografias, onde um usuário poderá acessar, visualizar as informações e efetuar o *download* do arquivo com a monografia. As inserções, edições e remoções estarão a cargo de um administrador.

Cada monografia conterà os metadados ilustrados na Figura 3.1:

```
{
  "Titulo" : data_string,      // Título da monografia
  "Autor" : data_string,      // Nome do autor
  "Orientador" : data_string, // Nome do orientador
  "Tipo" : data_string,       // Tipo de documento
  "Ano" : data_string,        // Ano da defesa
  "Semestre" : data_string,   // Semestre da defesa
  "resumo" : data_string,     // Resumo da monografia
}
```

Figura 3.1: Modelo documento dos metadados

Estes metadados são incluídos como um novo documento referente ao valor da chave “metadata” do documento padrão criado pelo MongoDB para suportar arquivos anexos. Isto pode ser visualizado na Figura 3.2.

```
{
  "id" : <unspecified>,      // ID único para cada documento
  "length" : data_number,    // Tamanho do arquivo em bytes
  "chunkSize" : data_number, // Tamanho de cada "parte" do arquivo
  "uploadDate" : data_date,  // Data do armazenamento do arquivo
  "md5" : data_string,       // Hash gerada a partir do arquivo
  "filename" : data_string,  // Nome do arquivo
  "metadata" : {             // Subdocumento com os metadados
    "Titulo" : data_string,  // Título da monografia
    "Autor" : data_string,   // Nome do autor
    "Orientador" : data_string, // Nome do orientador
    "Tipo" : data_string,    // Tipo de documento
    "Ano" : data_string,     // Ano da defesa
    "Semestre" : data_string, // Semestre da defesa
    "resumo" : data_string,  // Resumo da monografia
  }
}
```

Figura 3.2: Documento inserido na coleção “files”

Este documento da Figura 3.2 é inserido na coleção “files”. O MongoDB “quebra” o

arquivo anexo em várias partes e cada uma delas é inserida como um documento na coleção “*chunks*”, como pode ser visto na Figura 3.3.

```
{
  "_id" : <unspecified>,      // ID de cada parte do objeto arquivo
  "files_id" : <unspecified>, // ID do arquivo correspondente na coleção "files"
  "n" : chunk_number,        // Partes são numeradas em ordem, início em "0"
  "data" : data_binary,      // Dado armazenado em formato binário
}
```

Figura 3.3: Modelo documento na coleção “*chunks*”.

É importante ressaltar que este procedimento de “quebra”, ou *sharding*, é realizado automaticamente pelo MongoDB. Ao administrador cabe somente a edição dos metadados da Figura 3.1.

3.2 DIAGRAMA DE CASOS DE USO

Nesta seção é apresentado o diagrama de caso de uso do aplicativo a ser desenvolvido. A Figura 3.4 mostra o diagrama.

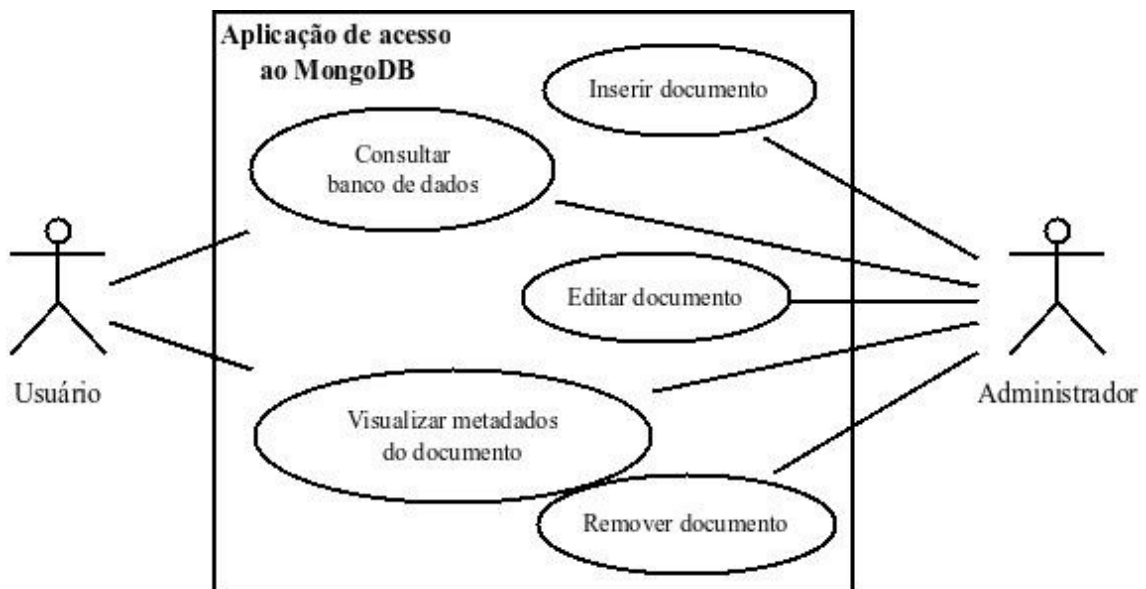


Figura 3.4: Diagrama de caso de uso da aplicação.

A aplicação é dividida em duas partes, possui a parte que é acessada por qualquer usuário cliente e a parte que somente o administrador tem acesso, mediante autenticação no

banco de dados.

Nas subseções a seguir são especificados os casos de uso referidos no diagrama da Figura 3.4. Para efetuar as tarefas como Usuário basta ter acesso sem autenticação, para acesso como Administrador é preciso autenticar-se na base de dados. Portanto, o servidor do banco de dados deve ter sido inicializado com opção de autenticação. Além disso, um usuário com permissão de escrita também deve ter sido criado anteriormente. Esses detalhes não fazem parte da implementação, portanto não serão detalhados.

3.2.1 Especificação do caso de uso Consultar banco de dados

Na tela inicial, é apresentado ao usuário a opção consultar o banco de dados, isto é, buscar registros (documentos) que foram adicionados anteriormente. Há uma caixa de texto para o usuário digitar uma palavra-chave para busca e uma caixa de seleção informando as opções disponíveis para a busca. É possível buscar pelo campo Autor, Orientador ou Título. Ao clicar no botão de buscar é gerada uma lista, disposta em uma tabela, dos registros encontrados. Na tabela consta apenas alguns metadados do documento (Título, Autor, Orientador e Ano). Neste momento, tem-se a diferenciação entre Usuário Cliente e Administrador. Caso esteja conectado como Usuário Cliente, estarão disponíveis as opções para visualização completa das informações dos metadados, através do botão “Detalhar”, e também a opção para descarregar o arquivo em PDF da monografia referida. Caso esteja autenticado como Administrador, estarão disponíveis as opções de visualizar os metadados, pelo botão “Detalhar”, editar metadados, através do botão “Editar”, e excluir o documento, através do botão “Excluir. Ainda na tela de Administrador, este poderá realizar uma consulta avançada através de um ícone que dará acesso a uma caixa de texto onde deverá ser escrita a consulta baseada em documento.

3.2.2 Especificação do caso de uso Visualizar metadados do documento

Fazendo uma busca com algum critério de pesquisa, gera-se uma lista com os documentos encontrados. Esta consulta pode ser realizada tanto a partir da tela de Usuário cliente quanto da tela de Administrador. A partir deste momento, tanto Usuário Cliente como Administrador, podem selecionar um elemento da lista e clicar no botão “Detalhar”.

Isso faz com que uma nova janela seja aberta, fornecendo as informações de todos os metadados do documento, incluindo Título, Autor, Orientador, Tipo, Ano, Semestre e Resumo.

3.2.3 Especificação do caso de uso Inserir documento

Após autenticado como Administrador, a tela mostra uma nova opção, que é a inserção de um novo documento. Isto está disponível através de um ícone com a legenda “Inserir Monografia”. Após clicar no ícone “Inserir Monografia” uma nova janela é aberta solicitando que os metadados da monografia sejam informados (Título, Autor, Orientador, Tipo, Ano, Semestre, Resumo) e, também, a opção para selecionar o arquivo PDF da monografia. Para não ocorrer a inserção de arquivos PDF iguais, é realizada uma verificação do valor *hash* gerado a partir do arquivo PDF. Efetua-se a busca para averiguar se este arquivo já está na base de dados.

Caso a inclusão ocorra com êxito, a janela é fechada e uma mensagem de sucesso é mostrada ao Administrador. Entretanto, ocorrendo um erro durante o procedimento de inserção, este é informado ao usuário para que examine as causas.

3.2.4 Especificação do caso de uso Alterar documento

Autenticado como Administrador, na tela de consulta, após a seleção de elemento da lista, clica-se no botão de “Editar”, uma nova janela será aberta, onde será possível alterar os metadados do documento.

Em cenário normal, os dados são enviados para a o banco e atualizados, entretanto, caso ocorra um erro durante o processo será exibida uma mensagem ao usuário.

3.2.5 Especificação do caso de uso Remover documento

É possível remover registros a partir da tela de Administrador. Após efetuar uma consulta, é possível selecionar um item listado e proceder a remoção através do botão “Excluir”. Havendo sucesso na remoção, é mostrada uma mensagem de operação realizada com sucesso. Caso ocorra algum erro na remoção, é exibida uma mensagem de alerta.

4 DESENVOLVIMENTO DA APLICAÇÃO

Neste capítulo é discutida a implementação da ferramenta de gerenciamento do repositório de monografias. Após isso, será feita uma análise do software desenvolvido.

A aplicação foi desenvolvida em Linguagem Java, utilizando o ambiente de desenvolvimento integrado Netbeans 7.1, com interface gráfica utilizando as ferramentas do *Swing*. Também foi utilizado o *driver* Java para conexão entre cliente e servidor mongo da base de dados.

A aplicação foi desenvolvida focando somente na parte do Cliente, entretanto, alguns pré-requisitos são necessários do lado do servidor de Banco de Dados. Entre estes, podemos destacar a inicialização do servidor com o parâmetro para possibilitar autenticação e a criação de um usuário e senha, com permissão de escrita no Banco de Dados onde serão armazenadas as monografias. Estas particularidades não serão detalhadas porque não fazem parte do escopo da aplicação.

O diagrama de classes apresentado na Figura 4.1 mostra a organização da aplicação. Os atributos foram suprimidos para simplificar o diagrama.

4.1 CLASSES IMPLEMENTADAS

Nesta seção é apresentado o diagrama de classes da aplicação desenvolvida.

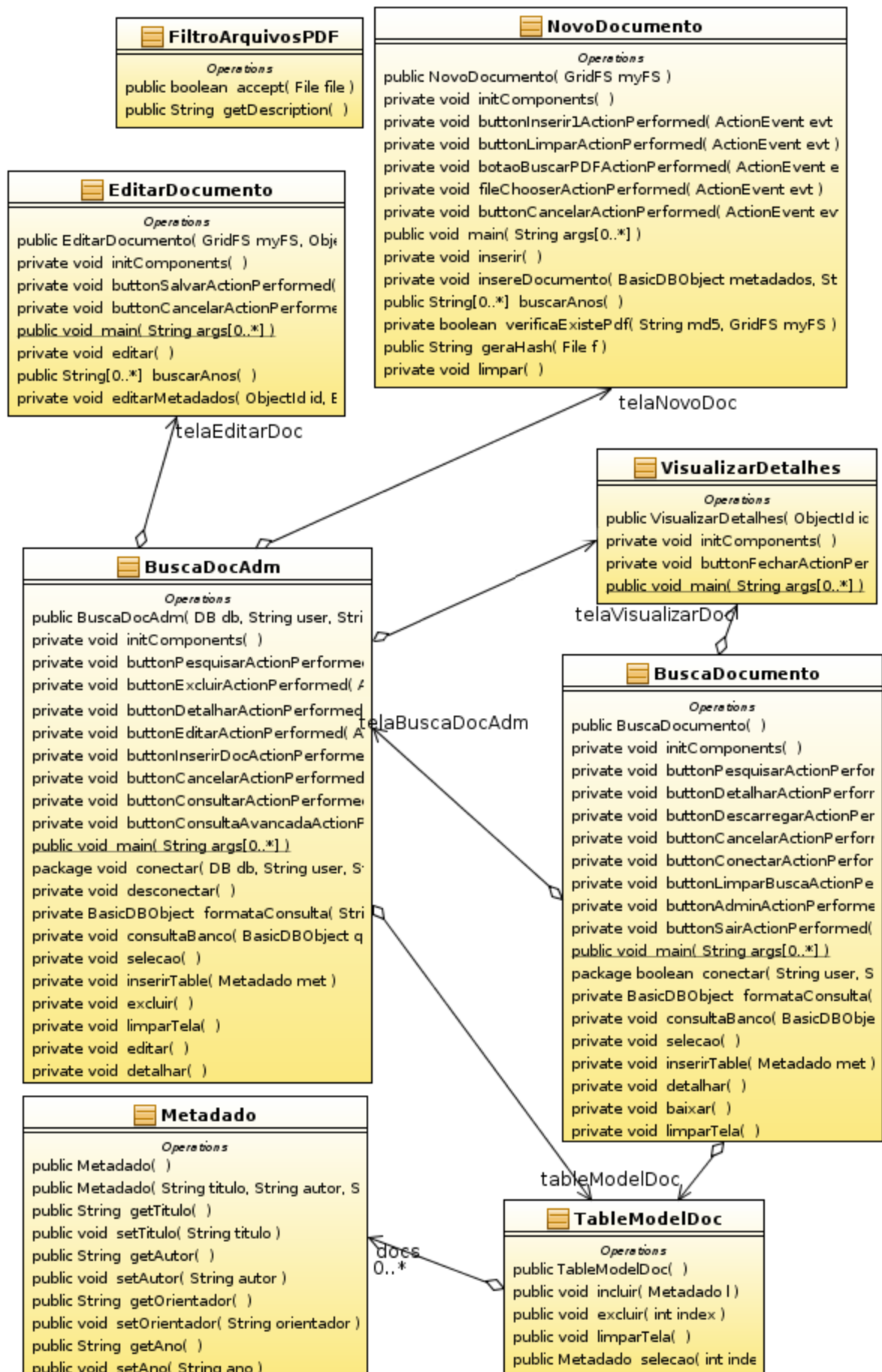


Figura 4.1: Diagrama de classes da aplicação.

Nas subseções a seguir são apresentadas as classes do diagrama de classes e suas principais funcionalidades.

4.1.1 Classe *BuscaDocumento* e Classe *BuscaDocAdm*

Estas classes implementam as interfaces da tela de Usuário Cliente e da tela principal do Administrador, respectivamente. Foram colocadas nesta mesma subseção porque são classes semelhantes e utilizam os mesmos métodos de consulta ao MongoDB.

O primeiro método a ser chamado na aplicação é o de conexão com o banco de dados, que pode ser visualizado na Figura 4.2. Através deste método é criada uma conexão com o banco que se mantém ativa durante o período de atividade no banco.

```
boolean conectar(String user, String pwd) {
    try {
        mongo = new Mongo(HOST, PORTA);
        db = mongo.getDB(DB_NAME);
        boolean aut = db.authenticate(user, pwd.toCharArray());
        if (aut) {
            myFS = new GridFS(db);
            return true;
        } else {
            return false;
        }
    }
    ...
}
```

Figura 4.2: Código do método conectar.

As consultas simples realizadas no painel de busca em qualquer uma das duas interfaces passam por um método para a formatação da consulta, isso porque a *query* a ser passada ao banco deve ter a sintaxe de documentos BSON. Na Figura 4.3 temos o código do método responsável pela correta formatação da consulta a ser enviada ao banco.

```
private BasicDBObject formataConsulta(String chave, String busca) {
    String exprReg = "^.*" + busca + ".*$";
    Pattern valor = Pattern.compile(exprReg, Pattern.CASE_INSENSITIVE);
    BasicDBObject query = new BasicDBObject("metadata." + chave, valor);
    return query;
}
```

Figura 4.3: Trecho do código do método formataConsulta.

O método “*formataConsulta*” recebe uma chave (Autor, Orientador ou Título) e um valor como parâmetros e o retorno é do tipo *BasicDBObject*, que é a instância no *driver* Java para a criação de um documento com sintaxe BSON. O retorno é então passado como parâmetro ao método “*consultaBanco*”, como pode ser visualizado na Figura 4.4.

O método “*consultaBanco*” recebe uma *query* baseada em documento para consulta no MongoDB. A consulta é realizada e se houver retorno, estes são inseridos na tabela de resultado da consulta, vistos nas interfaces de cada classe de busca.

Ao utilizar a opção de consulta avançada disponível na tela de interface de Administrador, a *query* é passada diretamente como parâmetro ao método “*consultaBanco*”, porque a sintaxe já está formato de documentos BSON.

```
private void consultaBanco(BasicDBObject query) {
    try (DBCursor cursor = myFS.getFileList(query)) {
        while (cursor.hasNext()) {
            BasicDBObject metadados = (BasicDBObject) cursor.next().get("metadata");
            String mTitulo = (String) metadados.get("Titulo");
            String mAutor = (String) metadados.get("Autor");
            String mOrientador = (String) metadados.get("Orientador");
            String mAno = (String) metadados.get("Ano");
            Metadado m = new Metadado(mTitulo, mAutor, mOrientador, mAno);
            inserirTable(m);
        }
    }
}
```

Figura 4.4: Código do método *consultaBanco*.

Na interface da tela de Usuário, estão disponíveis botões para pesquisar ou visualizar detalhes dos metadados das monografias e, também, o botão para descarregar a monografia. Já na interface da tela principal de Administrador estão disponíveis botões para editar metadados, inserir e excluir monografia e para realizar consultas avançadas.

O método para excluir monografias está implementado na classe “*BuscaDocAdm*”.

```
private void excluir() {
    ***
    if (selecionaOpcao == JOptionPane.OK_OPTION) {
        myFS.remove(id);
        JOptionPane.showConfirmDialog(null,
            " Exclusão efetuada com sucesso ", "Aviso!",
            JOptionPane.CLOSED_OPTION);
    }
}
```

Figura 4.5: Código do método *excluir*.

A Figura 4.5 mostra o código do método para exclusão. O método é simples, ele recebe o identificador de uma monografia selecionada e executa o método “*remove*” que está disponível na API do MongoDB.

4.1.2 Classe *NovoDocumento*

Esta classe implementa a interface da tela de inserção de monografias e os métodos utilizados para tal. As monografias a serem inseridas são obtidas através da captura dos metadados digitados na tela e da seleção do arquivo PDF de uma fonte local (disco rígido).

```
private void inserirDoc() {
    String textTitulo = labelTitulo.getText();
    String titulo = tfTitulo.getText();
    String textAutor = labelAutor.getText();
    String autor = tfAutor.getText();
    ...
    BasicDBObject metadados = new BasicDBObject();
    metadados.put(textTitulo, titulo);
    metadados.put(textAutor, autor);
    metadados.put(textOrientador, orientador);
    metadados.put(textTipoDoc, tipo);
    metadados.put(textAno, ano);
    metadados.put(textSemestre, semestre);
    metadados.put(textResumo, resumo);

    String pathArquivoPdf = getTfArquivoPdf().getText();
    try {
        GridFSInputFile inputFile = myFS.createFile(pathArquivoPdf);
        inputFile.setMetaData(metadados);
        inputFile.save();
    }
}
```

Figura 4.6: Código do método *insereDoc*.

Na Figura 4.6 têm-se o código do método responsável pela inserção da monografia. Os pares chave/valor são capturados a partir da interface e formatados para um documento de metadados. O arquivo PDF é selecionado e o MongoDB, através do método *createFile*, gera um documento específico com alguns campos padrões, como o visto na Figura 3.2.

O documento com os metadados capturados anteriormente é definido como valor da chave “*metadata*”, só então a monografia em si é guardada no Banco de dados através do método *save*.

4.1.3 Classe *EditarDocumento*

A classe “*EditarDocumento*” implementa a interface da tela de alteração dos metadados das monografias e o método responsável por isso.

Para editar os metadados de uma monografia armazenada no MongoDB é utilizado o método “*editarMetadados*”, que têm um trecho do seu código demonstrado na Figura 4.7.

```
private void editarMetadados (ObjectId id, BasicDBObject metadados) {
    GridFSDBFile file = myFS.findOne(id);
    file.setMetaData(metadados);
    file.save();
}
```

Figura 4.7: Código do método *editarMetadados*.

O método recebe a identificação do documento a ser editado e um novo documento de metadados atualizado, que será substituído no valor da chave “*metadata*”. No *driver* Java, há o método *findOne*, que retorna o documento correspondente ao valor passado como parâmetro. Após definir os novos metadados, é guardado novamente na base de dados.

4.1.4 Classe *VisualizarDetalhes*

A classe “*VisualizarDetalhes*” implementa a interface da tela para visualizar os metadados completos de uma monografia.

Nesta classe não foram criados métodos específicos, apenas foram gerados os métodos padrões (*get* e *set*) para setar os valores dos campos na inicialização dos componentes.

4.1.5 Classe *TableModelDoc* e Classe *Metadado*

As classes “*TableModelDoc*” e “*Metadado*” implementam, respectivamente, a interface da tabela de Resultado de Busca nas telas de Usuário Cliente e Administrador, e o tipo de dado que a compõe. Ainda na classe “*TableModelDoc*” são implementados os métodos para manipulação dos elementos na tabela, como incluir, excluir e selecionar. Estes

métodos não executam nenhuma ação no MongoDB, somente na tabela. Por exemplo, quando procedemos uma consulta na base de dados, o método incluir é chamado para que cada resultado seja incluído na tabela e mostrado para o usuário.

4.2 UTILIZAÇÃO DA APLICAÇÃO

A aplicação desenvolvida atende a dois tipos de usuários, o Usuário Cliente e o Usuário Administrador, permitindo ao usuário cliente buscar, visualizar metadados e descarregar a monografia, e ao Administrador consultar, visualizar os metadados, editar os metadados, remover e inserir monografias no Banco de Dados MongoDB.

Para seguir uma ordem de execução normal do aplicativo, primeiramente, será mostrada a utilização da aplicação para Usuário Cliente, que é a primeira tela a ser visualizada na aplicação. Posteriormente, será mostrado o uso a partir da tela principal de Administrador.

4.2.1 Utilização como Usuário Cliente

A primeira tela a ser exibida quando a aplicação é executada está exposta na Figura 4.8. Na figura, para melhor explicação, foram dispostos números de 1 a 8 para enumerar as ações disponíveis para o usuário. Estes números não se encontram na aplicação, somente na imagem.

Detalhando as ações, na sequência, o número “1” (ícone “Administrador”), indica a ação do usuário para autenticar-se como Administrador e ter acesso às opções avançadas. Após clicar no ícone “Administrador”, é solicitado nome de usuário e senha, anteriormente criados, como pode ser visualizado na Figura 4.9.

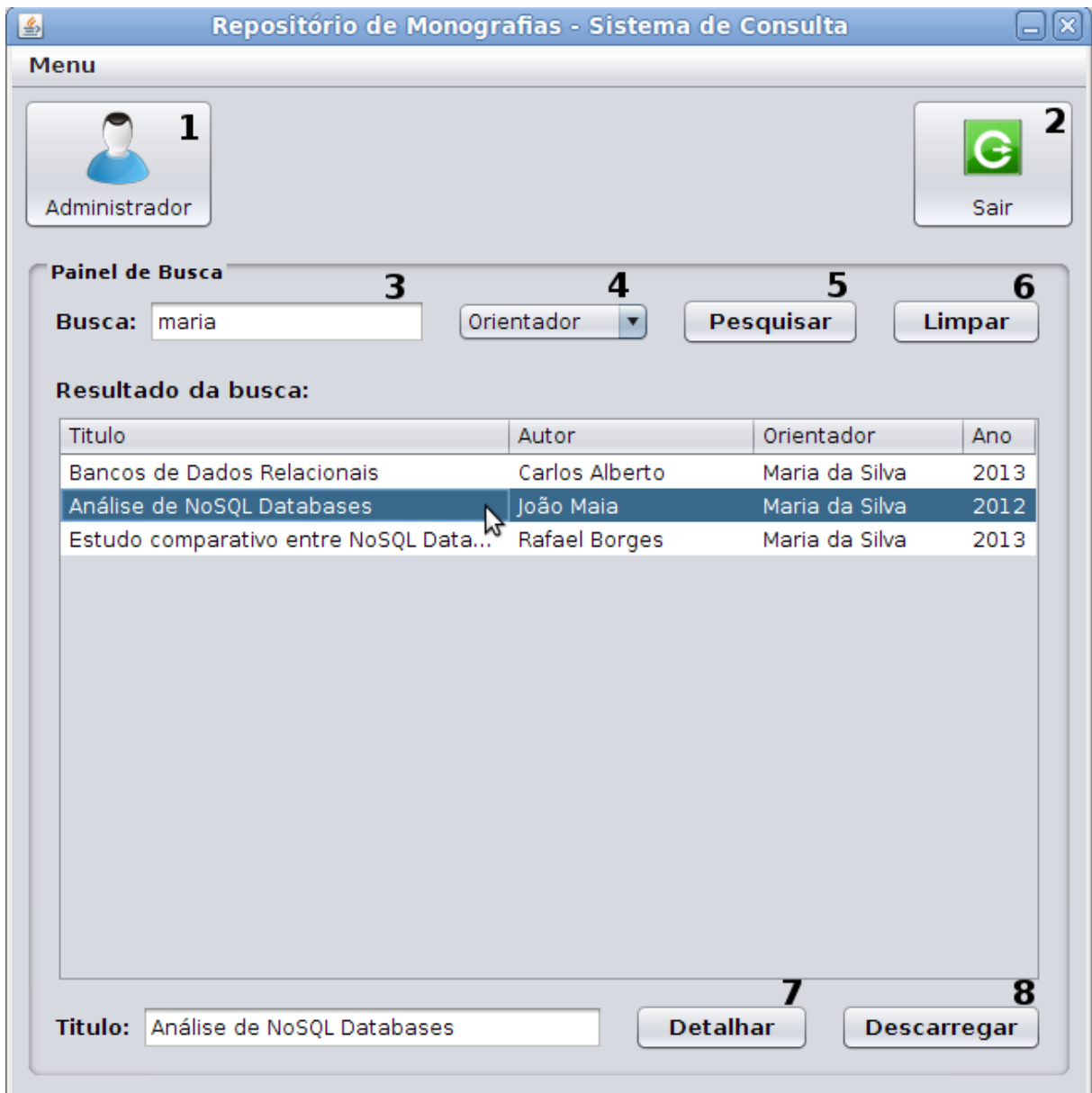


Figura 4.8: Tela principal Usuário Cliente.

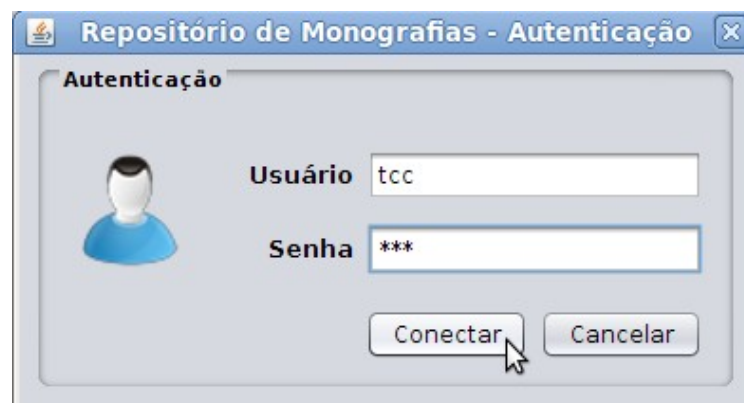


Figura 4.9: Tela de Autenticação como Administrador.

Seguindo com as ações do usuário da Figura 4.8, o número “2” indica a ação para o usuário sair do sistema. Já o número “3” mostra o campo onde é digitado o valor a ser buscado e, no número “4” é informado onde deseja buscar, se em Autor, Orientador ou Título. No exemplo da Figura 4.8 foi informado o valor “maria” a ser buscado na chave “Orientador”. Após informar os dados de busca, a consulta é realizada clicando-se no botão “Pesquisar”, referenciado pelo número “5”; isto faz com que uma tabela com os registros encontrados seja gerada abaixo do rótulo “Resultado da busca”. Lembrando que, para obter sucesso na busca, é necessário que monografias tenham sido inseridas anteriormente. Na próxima subseção é mostrado como inserir monografias. Caso o usuário queira buscar outro valor, basta repetir o procedimento a partir do número “3”, sendo também possível limpar a tela de busca clicando no botão “Limpar”, visto ao lado do número “6”.

Repositório de Monografias - Visualizar Detalhes

Visualizar Detalhes

Título

Autor

Orientador

Tipo **Ano** **Semestre**

Resumo

Figura 4.10: Tela de Visualização dos Detalhes.

As últimas ações disponíveis na tela de busca do usuário, são os botões “Detalhar”, referenciado pelo número “7”, e “Descarregar”, visto no número “8”. Para executar ambas ações, antes é preciso selecionar um item na tabela gerada pela busca. Feito isto, ao clicar em “Detalhar”, uma nova tela com todos os metadados da monografia para visualização e

leitura. A tabela resultante da consulta possui apenas alguns metadados; para a visualização completa dos detalhes é preciso seguir os passos mencionados. Na Figura 4.10 pode ser visualizada a tela gerada ao selecionar um item da tabela e clicar em “Detalhar”. Nesta tela, nenhuma alteração de metadados é possível, pois os campos estão desativados para edição.

Por último, ao selecionar um item da tabela resultante da busca e clicar no botão “Descarregar”, é possível fazer *download* da monografia selecionada, selecionando o local do disco onde se deseja armazenar a monografia em PDF.

4.2.2 Utilização como usuário Administrador

Após iniciar a execução da aplicação, e autenticar como Administrador, conforme indicam as Figuras 4.8 e 4.9, tem-se acesso a tela principal de Administrador com opções avançadas, que permitem incluir e excluir monografia, editar metadados e realizar consultas avançadas na base de dados. A Figura 4.11 mostra como é a tela de Administração.

Da mesma maneira do capítulo anterior, foram adicionados números (1 a 4) para melhor referenciar as ações que o administrador pode tomar. Algumas ações, como a busca simplificada por Autor, Orientador ou Título, e do botão “Detalhar”, são as mesmas que foram utilizadas no capítulo anterior, e funcionam da mesma maneira na tela de Administrador, portanto não serão explicadas novamente.

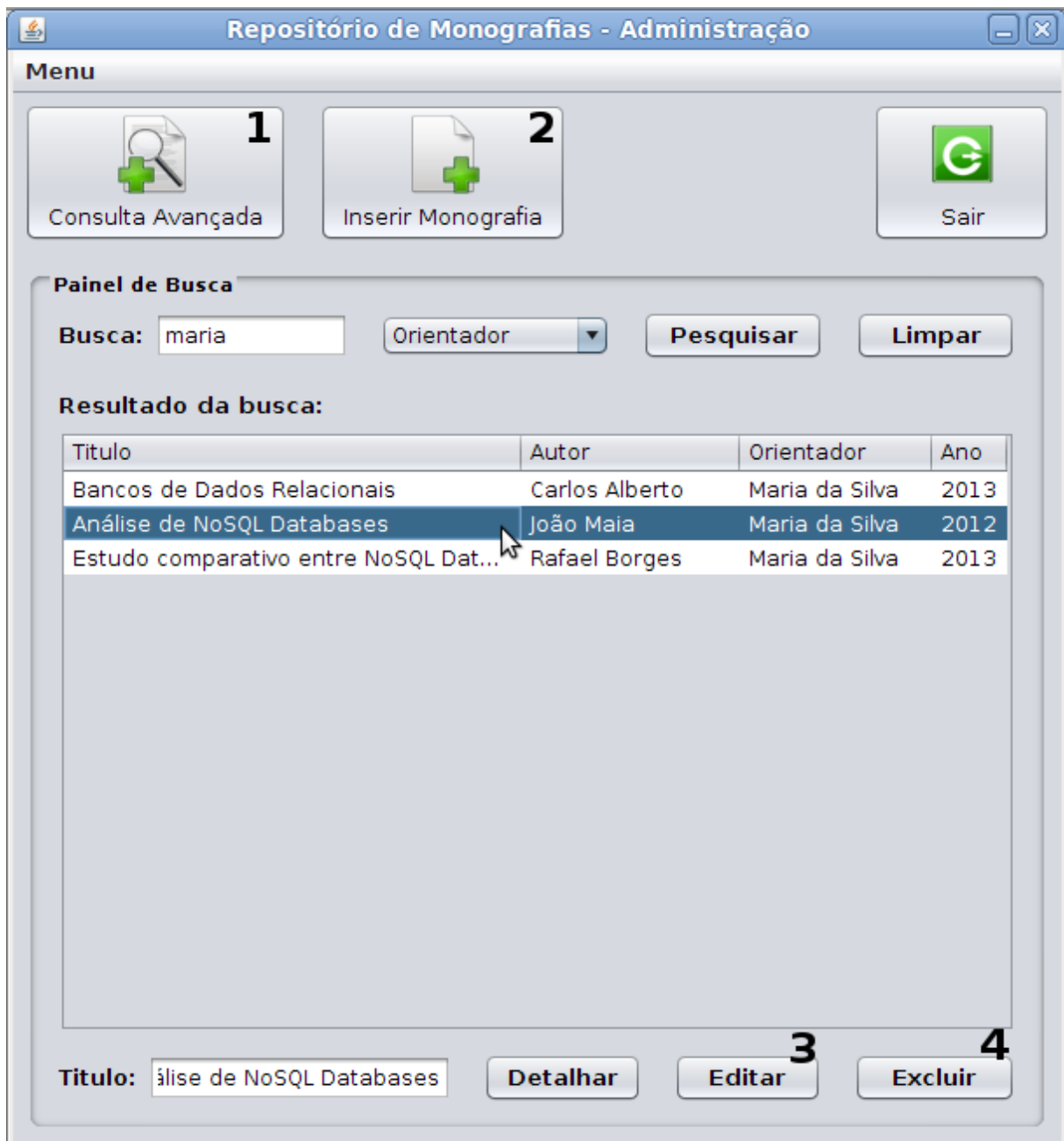


Figura 4.11: Tela principal do Administrador.

Os números adicionados à imagem indicam, em sequência, o ícone da ação para Consulta Avançada (1), o ícone de Inserção de Monografias (2), o botão para Edição dos Metadados (3) e o botão de Exclusão de Monografia (4). Nas próximas subseções são detalhados os resultados obtidos através de cada ação.

4.2.2.1 Inserção de monografias no MongoDB

Através da tela principal de Administrador, exposta na Figura 4.11, o Administrador pode clicar sobre o ícone Inserir Monografia, identificado naquela figura pelo número “2”. Feito isto, é exibida a tela de inserção de monografia, mostrada na Figura 4.12. Atente que na figura já constam os dados preenchidos para inserção, mas a tela é inicializada com os campos vazios.



A imagem mostra uma janela de software intitulada "Repositório de Monografias - Inserir Monografia". O formulário dentro da janela contém os seguintes campos e controles:

- Título:** Campo de texto com o valor "XML Databases".
- Autor:** Campo de texto com o valor "Anderson Araujo".
- Orientador:** Campo de texto com o valor "Maria da Silva".
- Tipo:** Menu suspenso com o valor "Monografia".
- Ano:** Menu suspenso com o valor "2010".
- Semestre:** Menu suspenso com o valor "1".
- Arquivo:** Campo de texto com o valor "e/ivan/Área de Trabalho/monografia55-xml_db.pdf" e um botão "Buscar" adjacente.
- Resumo:** Área de texto com o valor "Comparativo entre Bancos de Dados XML".
- Na base da janela, há três botões: "Limpar", "Inserir" (com o cursor do mouse sobre ele) e "Cancelar".

Figura 4.12: Tela de Inserção de Monografias.

Nesta tela, o administrador informa os metadados relativos à monografia a ser inserida, juntamente com o arquivo em formato PDF da monografia. Com os campos preenchidos, ao clicar no botão “Inserir” são gerados dois documentos, um relativo aos metadados (Título, Autor, Orientador, Tipo, Ano, Semestre e Resumo) e outro com algumas informações do arquivo PDF selecionado. O documento gerado para o arquivo PDF é uma propriedade do próprio MongoDB, através da ferramenta GridFS, já comentado

anteriormente neste trabalho. Para entender melhor, pode-se visualizar na Figura 4.13 como fica definido para o arquivo que estamos incluindo.

```
{
  "_id" : ObjectId("51109f4344ae5e55018130a3"),
  "chunkSize" : NumberLong(262144),
  "length" : NumberLong(3216204),
  "md5" : "0f114ae759c6f82d32cb59473869bebc",
  "filename" : "monografia55-xml_db.pdf",
  "contentType" : null,
  "uploadDate" : ISODate("2013-02-05T05:57:23.143Z"),
  "aliases" : null,
  "metadata" : { "Titulo" : "XML Databases",
    "Autor" : "Anderson Araujo",
    "Orientador" : "Maria da Silva",
    "Tipo" : "Monografia",
    "Ano" : "2010",
    "Semestre" : "1",
    "Resumo" : "Comparativo entre os Bancos de Dados XML." }
}
```

Figura 4.13: Documento gerado na inserção da monografia.

Existem dois pontos chaves na inserção, o documento com os metadados é inserido como valor na chave “metadata”, portanto é um documento dentro de outro documento. É importante ter conhecimento disso para as consultas avançadas que serão realizadas na sequencia. Outro ponto importante é a chave “md5”, que é única para cada arquivo PDF. Portanto, ao clicar no botão “Inserir” é chamado um método de verificação dessa chave “md5”, caso ela já conste na base de dados a monografia não será inserida, pois isso significa que o arquivo PDF já está na base de dados.

4.2.2.2 Execução de consultas avançadas no MongoDB

Após a inserção de monografias terem sido realizada com sucesso, é possível consultá-las através de busca simplificada como a realizada na subseção 4.2.1 tanto como usuário cliente ou como administrador. Ou, a partir da tela de administrador é possível a realização de consultas avançada baseadas em documento.

A partir da tela principal de Administrador, ao clicar no ícone Consulta Avançada, indicada pelo número “1” na Figura 4.11, é aberta uma caixa de texto como a visualizada na Figura 4.14, onde é possível escrever a consulta a ser realizada.

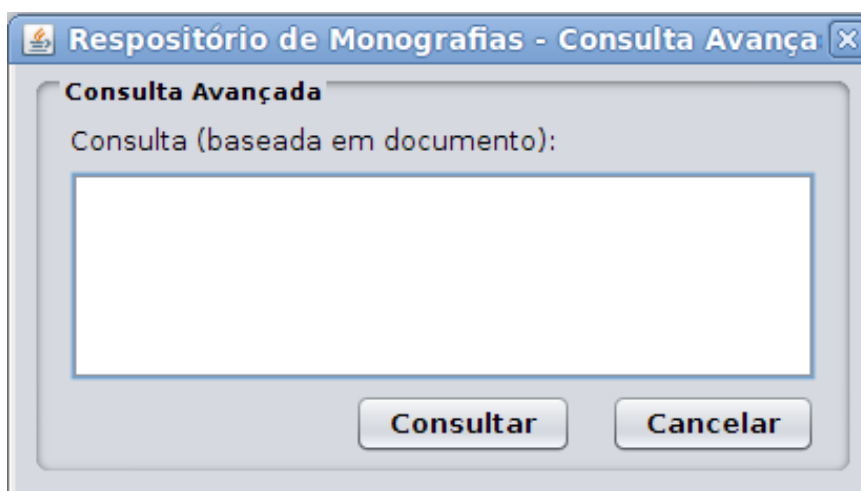


Figura 4.14: Caixa de texto para consulta avançada.

Para realizar a consulta, o administrador precisa recordar do modelo de documento e metadados que são construídos no momento de inserção da monografia. Portanto, como informado anteriormente, no documento gerado ao inserir uma monografia está presente uma chave chamada “metadata” contendo como valor um documento com os metadados. Então, as consultas usarão, na maioria das vezes, a sintaxe “**metadata.CHAVE_DOC_METADADO**”, onde **CHAVE_DOC_METADADO** é a chave contida no documento dos metadados (Título, Autor, Orientador, Tipo, Ano, Semestre ou Resumo).

Assim, as consultas seguem o modelo baseado em documento, exigindo a mesma sintaxe de um documento BSON, iniciando e fechando com o caractere chaves ({ }) e passando pelo menos um par chave/valor a ser buscado. No MongoDB estão disponíveis operadores especiais de consulta, como operadores de comparação, operadores lógicos, etc. No endereço <http://docs.mongodb.org/manual/reference/operators/#query-selectors> há uma lista dos operadores disponíveis e o significado de cada um deles.

A seguir são mostrados exemplos de algumas consultas que podem ser realizadas na base de dados através da aplicação desenvolvida.

- Consultar as monografias defendidas no ano de 2012:

```
{ "metadata.Ano" : "2012" }
```

- Buscar todas monografias defendidas no primeiro semestre, incluindo todos os anos.

{ "metadata.Semestre" : "1" }

- Consultar as monografias defendidas no ano de 2012, no segundo semestre:

{ "metadata.Ano": "2012", "metadata.Semestre": "2" }

A consulta anterior pode ser realizada de outra maneira, usando o operador \$and, desta forma, se a primeira expressão é avaliada como falsa, as expressões restantes não serão avaliadas:

{ \$and: [{"metadata.Ano" : "2012"}, {"metadata.Semestre": "2" }] }

- Consulta para retornar as monografias defendidas entre o ano de 2008 e 2011, incluindo os anos mencionados.

{ "metadata.Ano": { \$gte: "2008", \$lte: "2011" } }

- Consultar monografias realizadas antes de 2010:

{ "metadata.Ano": { \$lt: "2010" } }

- Buscar todas monografias defendidas , com exceção do ano de 2011.

{ "metadata.Ano": { \$ne: "2011" } }

- Consultar monografias realizadas antes de 2005 ou depois de 2010, excluindo estes.

{ \$or: [{"metadata.Ano" : { \$lt: "2005" } }, {"metadata.Ano": { \$gt:"2010" } }] }

- Buscar as monografias defendidas no ano de 2005, 2010 ou 2012, usando o operador \$in. Para isto é passado uma chave e uma lista de valores para consulta.

{ "metadata.Ano": { \$in: ["2005", "2010", "2012"] } }

Se trocarmos o operador \$in pelo operador \$nin e realizarmos a mesma consulta anterior, teremos como resultado todas as monografias, com exceção das defendidas no ano de 2005, 2010 e 2012.

{ "metadata.Ano": { \$nin: ["2005", "2010", "2012"] } }

- Consultar monografias orientadas por "Maria da Silva".

{ "metadata.Orientador" : "Maria da Silva" }

- Consultar monografias que tenham como Orientadora "Maria da Silva", no ano de 2012.

```
{"metadata.Orientador" : "Maria da Silva", "metadata.Ano" : "2012"}
```

- Consultar todas as monografias orientadas por "Maria da Silva" no ano de 2011, no primeiro semestre letivo:

```
{"metadata.Orientador" : "Maria da Silva", "metadata.Ano" : "2011",  
"metadata.Semestre" : "1"}
```

Caso não tenha certeza do nome completo do Orientador, é possível buscar por apenas um nome ou parte dele, isso é possível com o operador **\$regex** do *javascript*, que fornece capacidade para formar expressões regulares para a consulta.

```
{"metadata.Orientador": {$regex: ".*Maria"}}
```

É ideal que consultas utilizando **\$regex** sejam realizadas combinando o operador mencionado, com o operador **\$options** e sinalizador "i", para não haver diferenciação entre letras maiúsculas e minúsculas. Um exemplo para esta consulta.

```
{"metadata.Orientador" : {$regex: ".*maria", $options: "i" } }
```

- Consultar monografias que contenham a *string* "lucas" em Autor e primeiro semestre.

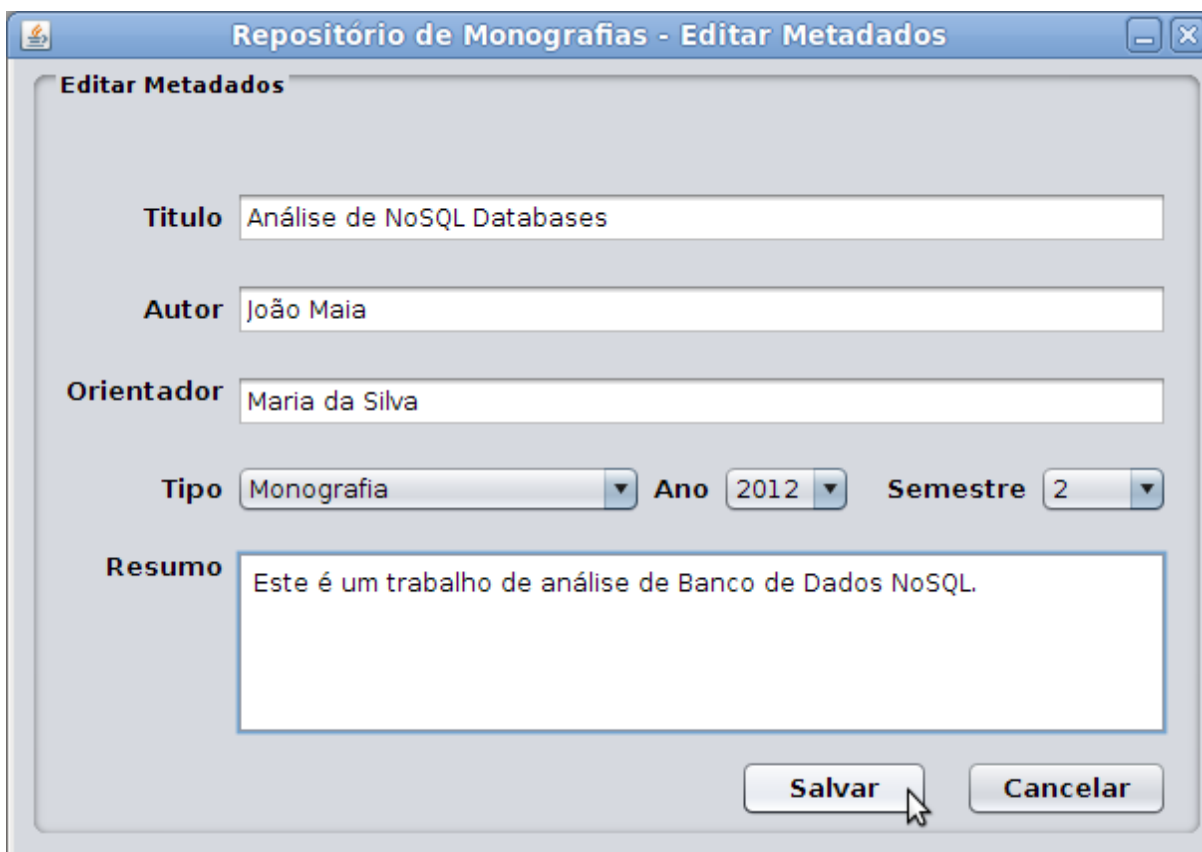
```
{ "metadata.Autor": { $regex: ".*lucas", $options: "i" }, "metadata.Semestre":  
"1"}
```

Os resultados encontrados são listados na tabela “Resultado da Busca”, onde é possível selecionar um elemento da tabela para visualizar, editar metadados ou excluir a monografia. O procedimento para estas ações será mostrado na próxima subseção.

4.2.2.3 Edição de Metadados de Monografias no MongoDB

Com as monografias já inseridas na base de dados, é possível fazer alterações nos metadados destas. A partir da tela principal de Administrador e após a realização de uma consulta aplicando os filtros necessários, os resultados gerados e mostrados na tabela de

busca estão disponíveis para alteração. Para proceder a edição, basta selecionar um item da tabela e clicar sobre o botão “Editar”, indicado pelo número “3” na Figura 4.11. Ao clicar neste botão, uma nova tela será aberta, como pode ser visualizado na Figura 4.15



The image shows a software window titled "Repositório de Monografias - Editar Metadados". The window contains a form with the following fields and values:

- Titulo:** Análise de NoSQL Databases
- Autor:** João Maia
- Orientador:** Maria da Silva
- Tipo:** Monografia
- Ano:** 2012
- Semestre:** 2
- Resumo:** Este é um trabalho de análise de Banco de Dados NoSQL.

At the bottom right of the form are two buttons: "Salvar" and "Cancelar".

Figura 4.15: Tela de Edição de Monografias.

Os metadados relativos à monografia selecionada são carregados automaticamente na tela de edição, basta editar quais campos forem necessários e clicar no botão “Salvar” para concluir a edição. Ao fazer isso, um novo documento com os metadados é criado que substituirá o documento antigo.

Importante ressaltar que o arquivo PDF da monografia não pode ser alterado, caso seja preciso alteração deste, o procedimento padrão a seguir é a exclusão da monografia e a inserção de uma monografia com o arquivo PDF. Nenhum campo pode estar vazio, se isso acontecer não será permitido salvar a edição.

4.2.2.4 Exclusão de monografias no MongoDB

O MongoDB facilita muito o processo de exclusão de documentos. Ao realizar uma consulta, a partir da tela principal do Administrador, os resultados são listados na tabela “Resultado da Consulta”, como mostra a Figura 4.11. É possível selecionar um item e proceder a exclusão através do botão “Excluir”, indicado naquela mesma figura pelo número “4”. Ao clicar neste botão é chamado o método para remover o documento, passando a identificação do documento a ser removido.

4.2.3 Tratamento de erros

Durante os testes foram identificados alguns erros que podem ser cometidos pelos usuários.

Partindo do princípio da execução aplicação, ao clicar no botão “Detalhar” ou “Descarregar” na tela de Usuário Cliente e não ter selecionado um item na tabela de resultados, um aviso será passado ao usuário sobre a necessidade de selecionar um elemento para detalhar ou descarregar. Outro possível erro pode ocorrer no momento da autenticação para acessar a tela de Administrador. Caso seja sejam digitados usuário ou senha incorretos uma mensagem de aviso será fornecida ao usuário. Outro aviso será fornecido ao usuário caso seja tentado realizar uma consulta passando menos de 3 caracteres, pois isto não é permitido.

Já na tela principal de Administrador, podem ocorrer erros ao tentar editar metadados ou excluir monografia sem a prévia seleção de um item da tabela. Um aviso será fornecido ao administrador para que ele primeiro selecione um elemento e somente então execute alguma ação.

Ainda na tela de administrador, ao tentar inserir uma nova monografia e não ter completado todos os metadados, será informado ao administrador para que isto seja corrigido, pois todos os campos são obrigatórios. Este mesmo erro pode acontecer na tela de edição de metadados, onde também é comunicado ao administrador para que complete as informações. Outro erro que pode acontecer ao inserir uma nova monografia é se o arquivo PDF informado já existe na base de dados. Antes da inserção é realizada uma verificação através da *hash* gerado a partir do arquivo, caso o arquivo já exista, não será possível inserir a monografia e o administrador será avisado para que corrija a situação.

4.3 TESTES DE DESEMPENHO

Nesta subseção é demonstrado um conjunto de testes de carga de dados que foram realizado, objetivando verificar a escalabilidade do banco e da aplicação na inserção de dados. Os testes baseiam-se na inserção de monografias e a captura do tempo para tal realização. Uma pequena modificação momentânea foi realizada na aplicação para permitir a inserção de monografias idênticas, isso foi feito para que ocorresse a inserção de monografias de mesmo tamanho e que a escrita dos dados no disco não proporcionasse uma falsa análise. Os testes foram realizados utilizando-se um computador portátil com Intel(R) Core(TM) i5 CPU M480 at 2.67GHz, com 4 processadores, 4GB de memória e rodando Ubuntu 10.04 de 32 bits.

A tabela 4.1 mostra os testes de inserções que foram realizados.

Nº de monografias inseridas	Tamanho PDF (Kilobytes)	Tempo (segundos)
1	1,5	0,002
	15	0,005
10	1,5	0,014
	15	0,028
100	1,5	0,085
	15	0,153
1.000	1,5	0,31
	15	0,659
10.000	1,5	2,69
	15	4,68
100.000	1,5	18
	15	43,42
500.000	1,5	72,5
1.000.000	1,5	134,6

Tabela 4.1: Carga de inserções na base dados.

A carga de dados da Tabela 4.1 busca ratificar os argumentos de que os bancos de dados NoSQL proporcionam bom desempenho (Strauch, 2011) . Nos testes foram inseridas monografias com arquivos PDF de tamanho 1,5KB e 15KB para mostrar que existe uma

grande diferença ao incluir a mesma quantidade de monografias, porém com arquivos PDF de tamanhos distintos. Esse fato fica evidente na tabela, onde mostra que o tempo para realizar a inserção da mesma quantidade de monografias foi praticamente o dobro (com uma pequena variação) para arquivos de tamanho 15KB em comparação com os arquivos de 1,5KB. Isso deve ser levado em consideração na análise de desempenho e também em comparações com outros SGBDs. Ainda sobre os testes, foram inseridas desde uma monografia, até 1.000.000 de monografias. As inserções de 500.000 monografias e 1.000.000 monografias foram realizadas somente com arquivos de 1,5KB pois estávamos trabalhando com sistema de 32 *bits* e o limite de 2GB era superado nos arquivos de 15KB, restrição esta detalhada na revisão de literatura do capítulo 2.2.6.

Um outro ponto que deve ser ressaltado é que o tempo consumido na inserção de grandes quantidades de documentos não tem aumento linear. Pode ser visto na mesma tabela que a inserção de 1.000.000 de arquivos de 1,5KB e seus metadados demora 134,6 segundos. Se esse tempo for dividido por 1.000 chega-se ao resultado de um tempo médio de 0,135 segundos para cada 1000 inserções, tempo este bem abaixo dos 0,31 segundos registrados na inserção de 1000 documentos realizados na carga anterior.

Além dos pontos ressaltados, outra questão importante é a escalabilidade horizontal que pode ser realizada para suprir deficiências de hardware como neste caso, onde a escrita em disco comprovadamente é um problema. No caso dos testes realizados, se estivéssemos trabalhando com outros servidores e utilizando várias instâncias do MongoDB a tarefa de escrita em disco seria dividida entre estes servidores, diminuindo assim o tempo gasto para escrita de dados. Não foram efetuados testes nesse sentido, mas as referências bibliográficas apontam que o ganho de desempenho é considerável ao utilizar múltiplas instâncias do Mongo. O MongoDB fornece nativamente o recurso de *sharding* automático, muito útil para escalabilidade horizontal, onde os arquivos são divididos em partes pequenas e podem ser armazenadas em um ou mais servidores.

4.4 DOWNLOAD DA APLICAÇÃO

Os códigos fonte da aplicação e o arquivo para instalação do MongoDB estão disponíveis em <http://www.inf.ufsm.br/~ivanf/MongoDB>. Antes de executar a aplicação é recomendável conferir se o Java está instalado no computador e qual a versão. A aplicação foi desenvolvida utilizando o Java 7, só funcionará a partir desta versão.

5 CONCLUSÕES E TRABALHOS FUTUROS

A finalidade deste trabalho foi fornecer uma visão geral e introduzir ao movimento dos NoSQLs *Databases*, que surgiu com força nos últimos anos, sendo uma alternativa aos SGBD predominantemente relacionais. Para isto, um dos objetivos foi analisar alguns bancos de dados NoSQL disponíveis para utilização e que já tinham uma certa popularidade. Foram analisados os bancos Redis, Riak, Cassandra, Hbase, CouchDB, MongoDB e Neo4j. Através da análise foi possível obter embasamento para a necessidade de trabalhar com uma base de dados não relacional.

O estudo desenvolvido sobre os NoSQL *Databases* teve a intenção de adquirir conhecimentos sobre como cada um deles funciona, de que forma os dados são armazenados, qual a arquitetura que cada um deles utiliza, quais os requisitos para o funcionamento, se são proprietários ou *open source* e também para compreender e propagar tecnologias que são conceitualmente distintas das tradicionais e que estão em grande crescimento, com o intuito de proporcionar alternativas de escolha e fundamentação no momento de optar como os dados de um sistema serão armazenados.

Após a realização dos estudos sobre os NoSQL *Databases*, um deles foi escolhido, levando em consideração as informações obtidas na análise individual, para o desenvolvimento de uma aplicação. O Banco de Dados escolhido foi o MongoDB, e as razões que levaram à escolha do referido banco foram discutidas no capítulo 3 deste trabalho. Após a escolha do banco, foi proposto o desenvolvimento da aplicação para a manipulação dos dados armazenados neste banco de dados.

A aplicação proposta foi desenvolvida na linguagem de programação Java e o principal objetivo dela é gerenciar o conteúdo de uma base de dados. Com a utilização desta

aplicação, é possível armazenar, na base de dados do MongoDB, arquivos de monografias, incluindo seus metadados, em forma de documentos BSON, também é possível editá-los, removê-los e consultá-los. Nos exemplos e nos testes da aplicação foram utilizados documentos que procuraram ser fidedignos aos metadados encontrados nas monografias dos cursos de graduação atuais.

Como sugestão para trabalhos futuros, indica-se a transcrição da aplicação desenvolvida neste trabalho para alguma linguagem de programação para *web*, tornando a aplicação mais dinâmica, que vem a ser um dos objetivos dos bancos NoSQL. Outra sugestão é que a aplicação possa permitir a inclusão de campos extras, se necessário, nos metadados. Por exemplo, uma monografia pode ter, além do orientador, um co-orientador, enquanto outras não monografias continuariam não necessitando deste campo.

Outro ponto que pode ser incluído é o cadastro de novos usuários administradores a partir da tela de Administrador da aplicação.

Por último, sugere-se efetuar a utilização de vários servidores para distribuição e replicação dos dados, efetuando cargas de dados, consulta, edição e exclusão para testes de desempenho a medida que são agregados novos servidores.

REFERÊNCIAS

ANDERSON, J. C.; Lehnardt, J.; Slater, N. **CouchDB: The Definitive Guide**. Ed. O'Reilly Media, 2010.

BSON. **Binary JSON**. Disponível em <http://bsonspec.org/>. Acesso em novembro de 2012.

Campanelli, P. **The architecture of REDIS**. Disponível em <http://www.enjoythearchitecture.com/redis-architecture.html>. Acesso em novembro de 2012.

Cassandra. **The Apache Cassandra database**. Disponível em <http://cassandra.apache.org>. Acesso em dezembro de 2012.

Cho, T. **Apache Cassandra Quick tour**. Disponível em <http://javamaster.wordpress.com/2010/03/22/apache-cassandra-quick-tour/>. Acesso em dezembro de 2012.

CouchDB. **Apache CouchDB**. Disponível em <http://couchdb.apache.org/>. Acesso em novembro de 2012.

Datastax. **Apache Cassandra Documentation**. Disponível em <http://www.datastax.com/docs/1.1/index>. Acesso em dezembro de 2012.

FindTheBest. **Compare NoSQL Databases**. Disponível em <http://nosql.findthebest.com/>. Acesso em novembro de 2012.

Hbase. **Apache Hbase**. Disponível em <http://hbase.apache.org/>. Acesso em dezembro de 2012.

HEWITT, Eben. **Cassandra: The Definitive Guide**. Sebastopol: O'Reilly Media, 2010.

Ho, R. **MongoDb Architecture**. Disponível em <http://horicky.blogspot.com.br/2012/04/mongodb-architecture.html>. Acesso em novembro de 2012.

Ippolito, Bob. **Drop ACID and think about Data**. March 2009. Disponível em <http://blip.tv/file/1949416/>. Acesso em novembro de 2012.

JSON. **Introducing JSON**. Disponível em <http://www.json.org/>. Acesso em novembro de 2012.

Katz, D. **CouchDb Architecture**. Disponível em http://damienkatz.net/2005/04/couchdb_archite.html. Acesso em novembro de 2012.

Leavitt, Neal. Will NoSQL Databases Live Up to Their Promise?. **Computer. IEEE**, New York,, vol. 43, n. 2, p.12-14, Fev. 2010.

MongoDB. **MongoDB Docs**. Disponível em <http://www.mongodb.org/display/DOCS/Home>. Acesso em novembro de 2012.

Neo4j. **The Neo4j Manual**. Disponível em <http://docs.neo4j.org/chunked/stable/>. Acesso em dezembro de 2012.

Orend, K. **Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer**. Master Thesis, Technical University of Munich, Munich, 2010.

Popescu, A. **Presentation: NoSQL at CodeMash - An Interesting NoSQL categorization**. Disponível em <http://nosql.mypopescu.com/post/396337069/presentation-nosql-codemash-an-interesting-nosql>. Acesso em novembro de 2012.

Pritchett, Dan. BASE: An Acid Alternative. **Queue**. ACM, New York, vol. 6, n. 3, p.48-55, mai/jun. 2008.

Redmond, E.; Wilson, J. R. **Seven Databases in seven weeks - A Guide to Modern Databases and the NoSQL Movement**. Ed. The Pragmatic Programmers, 2012.

Riak. **Riak Basho**. Disponível em <http://riak.basho.com/>. Acesso em novembro de 2012.

Strauch, C. **NoSQL Databases**. Disponível em <http://www.christof-strauch.de/nosql dbs.pdf>. Acesso em novembro de 2012.

Seguin K. **The Little Redis Book**. Disponível em <https://github.com/karlseguin/the-little-redis-book>. Acesso em novembro de 2012.

Tiwari, S. **Professional NoSQL**. Ed. Wrox Programmer to Programmer, 2011.

Wikipedia. **JSON**. Disponível em <http://en.wikipedia.org/wiki/JSON>. Acesso em novembro de 2012.

APÊNDICES

APÊNDICE A – Código fonte da aplicação

A.1 BuscaDocumento.java

```
package aplicacao;
import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCursor;
import com.mongodb.Mongo;
import com.mongodb.gridfs.GridFS;
import com.mongodb.gridfs.GridFSDBFile;
import java.io.File;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.regex.Pattern;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.JTable;
import org.bson.types.ObjectId;

public class BuscaDocumento extends javax.swing.JFrame {

    static final String HOST = "localhost";
    static final int PORTA = 27017;
    static final String DB_NAME = "repositorio";
    private Mongo mongo;
    private DB db;
    private GridFS myFS;
```

```

private TableModelDoc tableModelDoc;
BuscaDocAdm telaBuscaDocAdm;

public BuscaDocumento() {
    initComponents();

    tableModelDoc = new TableModelDoc();
    tableDoc.setModel(tableModelDoc);
    tableDoc.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
    tableDoc.getColumnModel().getColumn(0).setPreferredWidth(tableDoc.getWidth() / 2);
    tableDoc.getColumnModel().getColumn(1).setPreferredWidth(tableDoc.getWidth() / 4);
    tableDoc.getColumnModel().getColumn(2).setPreferredWidth(tableDoc.getWidth() / 4);
    tableDoc.getColumnModel().getColumn(3).setPreferredWidth(tableDoc.getWidth() / 20);
    tableModelDoc.fireTableDataChanged();
    conectar(user, pwd);
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">

// </editor-fold>

private void buttonPesquisarActionPerformed(java.awt.event.ActionEvent evt) {

    limparTela();
    String busca = tfBusca.getText();
    if (busca.length() > 2) {
        String buscaCampo = (String) comboBusca.getSelectedItem();
        BasicDBObject query = formataConsulta(buscaCampo, busca);
        consultaBanco(query);
    } else {
        JOptionPane.showConfirmDialog(null,
            " Digite uma busca com pelo menos 3 caracteres! ", "Alerta!",
            JOptionPane.CLOSED_OPTION);
    }
}

private void menuSUActionPerformed(java.awt.event.ActionEvent evt) {

    jdLogin.pack();
    jdLogin.setVisible(true);
}
}

```

```

private void buttonDetalharActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:
    detalhar();
}

private void tableDocMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    selecao();
}

private void buttonDescarregarActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:
    try {
        baixar();
    } catch (IOException ex) {
        Logger.getLogger(BuscaDocumento.class.getName()).log(Level.SEVERE,
null, ex);
    }
}

private void menuSairActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    System.exit(0);
}

private void tableDocKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    selecao();
}

private void buttonCancelarActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:
    jdLogin.setVisible(false);
}

private void buttonConectarActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:

    user = tfUsuario.getText();
    pwd = new String(pfSenha.getPassword());
    boolean autenticacao = conectar(user, pwd);
    if (autenticacao) {
        telaBuscaDocAdm = new BuscaDocAdm(db, user, pwd);
        telaBuscaDocAdm.setVisible(true);
        jdLogin.setVisible(false);
        dispose();
    } else {
        JOptionPane.showMessageDialog(null, "Login incorreto!", "Alerta",
JOptionPane.WARNING_MESSAGE);
    }
}

private void buttonLimparBuscaActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:

```

```

        tfBusca.setText("");
        limparTela();
    }

    private void jmLimparActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        tfBusca.setText("");
        limparTela();
    }

    private void buttonAdminActionPerformed(java.awt.event.ActionEvent evt)
{
        // TODO add your handling code here:
        jdLogin.pack();
        jdLogin.setVisible(true);
    }

    private void buttonSairActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        System.exit(0);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {

        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting
code (optional) ">
        //</editor-fold>

        /*
         * Create and display the form
         */
        java.awt.EventQueue.invokeLater(new Runnable() {

            public void run() {
                new BuscaDocumento().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton buttonAdmin;
    private javax.swing.JButton buttonCancelar;
    private javax.swing.JButton buttonConectar;
    private javax.swing.JButton buttonDescarregar;
    private javax.swing.JButton buttonDetalhar;
    private javax.swing.JButton buttonLimparBusca;
    private javax.swing.JButton buttonPesquisar;
    private javax.swing.JButton buttonSair;
    private javax.swing.JComboBox comboBusca;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JDialog jdLogin;

```

```

private javax.swing.JLabel jlImgAut;
private javax.swing.JMenuItem jmLimpar;
private javax.swing.JPopupMenu.Separator jsSeparador1;
private javax.swing.JPopupMenu.Separator jsSeparador2;
private javax.swing.JLabel labelResultadoBusca;
private javax.swing.JLabel labelTitulo;
private javax.swing.JMenu menuMenu;
private javax.swing.JMenuItem menuSU;
private javax.swing.JMenuItem menuSair;
private javax.swing.JPasswordField pfSenha;
private javax.swing.JTable tableDoc;
private javax.swing.JTextField tfBusca;
private javax.swing.JTextField tfTituloDoc;
private javax.swing.JTextField tfUsuario;
// End of variables declaration

public javax.swing.JTable getTableDoc() {
    return tableDoc;
}

public javax.swing.JTextField getTfTituloDoc() {
    return tfTituloDoc;
}

public void setTfTituloDoc(javax.swing.JTextField tfTituloDoc) {
    this.tfTituloDoc = tfTituloDoc;
}

boolean conectar(String user, String pwd) {
    try {
        mongo = new Mongo(HOST, PORTA);
        db = mongo.getDB(DB_NAME);
        boolean aut = db.authenticate(user, pwd.toCharArray());
        if (aut) {
            myFS = new GridFS(db);
            return true;
        } else {
            return false;
        }
    } catch (Exception e) {
        JOptionPane.showConfirmDialog(null,
            "Não foi possível conectar-se ao servidor: " + HOST, "Alerta!",
            JOptionPane.CLOSED_OPTION);
        System.exit(0);
    }
    return false;
}

// metodo para formatar a string da consulta a ser realizada
private BasicDBObject formataConsulta(String chave, String busca) {
    String exprReg = "^.*" + busca + ".*$";
    Pattern valor = Pattern.compile(exprReg, Pattern.CASE_INSENSITIVE);
    BasicDBObject query = new BasicDBObject("metadata." + chave, valor);
    return query;
}

// metodo para consultar o banco de dados,
// retornando uma lista de registros

```

```

private void consultaBanco(BasicDBObject query) {
    try {
        DBCursor cursor = myFS.getFileList(query).sort(query);
        while (cursor.hasNext()) {
            BasicDBObject metadados = (BasicDBObject)
cursor.next().get("metadata");
            String mTitulo = (String) metadados.get("Titulo");
            String mAutor = (String) metadados.get("Autor");
            String mOrientador = (String) metadados.get("Orientador");
            String mAno = (String) metadados.get("Ano");
            Metadado m = new Metadado(mTitulo, mAutor, mOrientador, mAno);
            inserirTable(m);
        }
    } catch (Exception e) {
        System.out.println("Sem resultado...");
    }
}

//metodo para selecionar
private void selecao() {
    int index = getTableDoc().getSelectedRow();
    if (index == -1) {
        return;
    }
    Metadado m = tableModelDoc.selecao(index);
    getTfTituloDoc().setText(m.getTitulo());
}

//metodo para inserir o registro na tabela de consulta
private void inserirTable(Metadado met) {

    if (met != null) {
        tableModelDoc.incluir(met);
    }
}

//metodo para chamar uma nova tela para visualizar os detalhes
private void detalhar() {
    int index = getTableDoc().getSelectedRow();
    System.out.println("index: " + index);
    if (index == -1) {
        JOptionPane.showConfirmDialog(null,
            " Nenhum registro selecionado! ", "Alerta!",
            JOptionPane.CLOSED_OPTION);
        return;
    }
    Metadado m = tableModelDoc.selecao(index);
    BasicDBObject query = new BasicDBObject("metadata.Titulo",
m.getTitulo());
    GridFSDBFile fileout = myFS.findOne(query);
    ObjectId id = (ObjectId) fileout.getId();

    BasicDBObject metadados = (BasicDBObject) fileout.get("metadata");
    String mTitulo = (String) metadados.get("Titulo");
    String mAutor = (String) metadados.get("Autor");
    String mOrientador = (String) metadados.get("Orientador");
    String mTipo = (String) metadados.get("Tipo");
    String mAno = (String) metadados.get("Ano");
}

```

```

String mSemestre = (String) metadados.get("Semestre");
String mResumo = (String) metadados.getString("Resumo");

VisualizarDetalhes telaVisualizarDoc = new VisualizarDetalhes(id,
mTitulo, mAutor, mOrientador, mTipo, mAno, mSemestre, mResumo);
telaVisualizarDoc.setVisible(true);
}

// metodo para fazer o download do arquivo PDF
private void baixar() throws IOException {

    int index = getTableDoc().getSelectedRow();
    if (index == -1) {
        JOptionPane.showConfirmDialog(null,
            " Nenhum registro selecionado! ", "Alerta!",
            JOptionPane.CLOSED_OPTION);
        return;
    }
    Metadado m = tableModelDoc.selecao(index);
    BasicDBObject id = new BasicDBObject("metadata.Titulo",
m.getTitulo());

    GridFSDBFile fileout = myFS.findOne(id);
    JFileChooser salvandoArquivo = new JFileChooser();
    String nome = fileout.getFilename();
    salvandoArquivo.setSelectedFile(new File(nome));
    int resultado = salvandoArquivo.showSaveDialog(this);

    if (resultado == JFileChooser.APPROVE_OPTION) {
        File salvarArquivoEscolhido = salvandoArquivo.getSelectedFile();
        if (salvarArquivoEscolhido.exists() == true) {

            int selecionaOpcao = JOptionPane.showConfirmDialog(null,
                " O arquivo já existe, deseja sobrescreve-lo? ", null,
                JOptionPane.OK_CANCEL_OPTION);
            if (selecionaOpcao == JOptionPane.OK_OPTION) {
                fileout.writeTo(salvarArquivoEscolhido);
            }
        }
        else {
            fileout.writeTo(salvarArquivoEscolhido);

            JOptionPane.showConfirmDialog(null,
                " Arquivo descarregado com sucesso ", null,
                JOptionPane.CLOSED_OPTION);
        }
    }
}

// Metodo para limpar a tela de busca
private void limparTela() {

    int x = tableModelDoc.getRowCount();
    for (int i = 0; i < x; i++) {
        tableModelDoc.limparTela();
    }
    tfTituloDoc.setText("");
}
}

```


A.2 BuscaDocAdm.java

```

package aplicacao;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCursor;
import com.mongodb.Mongo;
import com.mongodb.gridfs.GridFS;
import com.mongodb.gridfs.GridFSDBFile;
import com.mongodb.util.JSON;
import com.mongodb.util.JSONParseException;
import java.net.UnknownHostException;
import java.util.regex.Pattern;
import javax.swing.JOptionPane;
import javax.swing.JTable;
import org.bson.types.ObjectId;

public class BuscaDocAdm extends javax.swing.JFrame {

    private DB db;
    Mongo mongo;
    GridFS myFS;
    private String user;
    private String pwd;
    private TableModelDoc tableModelDoc;

    public BuscaDocAdm(DB db, String user, String pwd) {
        this.db = db;
        this.user = user;
        this.pwd = pwd;
        initComponents();

        //Adicionado para gerar tabela
        tableModelDoc = new TableModelDoc();
        tableDoc.setModel(tableModelDoc);
        tableDoc.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        tableDoc.getColumnModel().getColumn(0).setPreferredWidth(tableDoc.getWidth() / 2);
        tableDoc.getColumnModel().getColumn(1).setPreferredWidth(tableDoc.getWidth() / 4);
        tableDoc.getColumnModel().getColumn(2).setPreferredWidth(tableDoc.getWidth() / 4);
        tableDoc.getColumnModel().getColumn(3).setPreferredWidth(tableDoc.getWidth() / 20);
        tableModelDoc.fireTableDataChanged();
        conectar(db, user, pwd);
    }

    /**
     * This method is called from within the constructor to initialize the
     form.

```

```

    * WARNING: Do NOT modify this code. The content of this method is
always
    * regenerated by the Form Editor.
    */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">

// </editor-fold>

    private void buttonPesquisarActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:
    limparTela();
    String busca = tfBusca.getText();
    if (busca.length() > 2) {
        String buscaCampo = (String) comboBusca.getSelectedItem();
        BasicDBObject query = formataConsulta(buscaCampo, busca);
        consultaBanco(query);
    } else {
        JOptionPane.showConfirmDialog(null,
            " Digite uma busca com pelo menos 3 caracteres! ", "Alerta!",
            JOptionPane.CLOSED_OPTION);
    }
}

    private void buttonExcluirActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:tfIdObjeto
    excluir();
}

    private void buttonDetalharActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:
    //Codigo para Visualizar detalhes
    detalhar();
}

    private void jmSairActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    System.exit(0);
}

    private void
jmInserirMonografiaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    NovoDocumento telaNovoDoc = new NovoDocumento(myFS);
    telaNovoDoc.setVisible(true);
}

    private void tableDocMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    selecao();
}

    private void buttonEditarActionPerformed(java.awt.event.ActionEvent evt)
{

```

```

        // TODO add your handling code here:
        editar();
    }

    private void tableDocKeyReleased(java.awt.event.KeyEvent evt) {
        // TODO add your handling code here:
        selecao();
    }

    private void buttonSairActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        System.exit(0);
    }

    private void buttonLimparBuscaActionPerformed(java.awt.event.ActionEvent
    evt) {
        // TODO add your handling code here:
        tfBusca.setText("");
        limparTela();
    }

    private void buttonInserirDocActionPerformed(java.awt.event.ActionEvent
    evt) {
        // TODO add your handling code here:
        NovoDocumento telaNovoDoc = new NovoDocumento(myFS);
        telaNovoDoc.setVisible(true);
    }

    private void buttonCancelarActionPerformed(java.awt.event.ActionEvent
    evt) {
        // TODO add your handling code here:
        jdConsultaAvancada.setVisible(false);
    }

    private void buttonConsultarActionPerformed(java.awt.event.ActionEvent
    evt) {
        // TODO add your handling code here:
        limparTela();
        String query = textConsultaAvancada.getText();
        if (query.length() > 3) {
            try {
                BasicDBObject n = (BasicDBObject) JSON.parse(query);
                consultaBanco(n);
                jdConsultaAvancada.setVisible(false);
            } catch (JSONException evParseException) {
                jdConsultaAvancada.setVisible(false);
            }
        } else {
            jdConsultaAvancada.setVisible(false);
        }
    }

    private void
    buttonConsultaAvancadaActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        jdConsultaAvancada.pack();
        jdConsultaAvancada.setVisible(true);
    }

```

```

private void
jmConsultaAvancadaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    jdConsultaAvancada.pack();
    jdConsultaAvancada.setVisible(true);
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {

    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting
code (optional) ">
    //</editor-fold>

    /**
     * Create and display the form
     */
    java.awt.EventQueue.invokeLater(new Runnable() {

        public void run() {
            new BuscaDocAdm(null, null, null).setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton buttonCancelar;
private javax.swing.JButton buttonConsultaAvancada;
private javax.swing.JButton buttonConsultar;
private javax.swing.JButton buttonDetalhar;
private javax.swing.JButton buttonEditar;
private javax.swing.JButton buttonExcluir;
private javax.swing.JButton buttonInserirDoc;
private javax.swing.JButton buttonLimparBusca;
private javax.swing.JButton buttonPesquisar;
private javax.swing.JButton buttonSair;
private javax.swing.JComboBox comboBusca;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JMenuBar jMenuBar1;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JDialog jdConsultaAvancada;
private javax.swing.JMenuItem jmConsultaAvancada;
private javax.swing.JMenuItem jmInserirMonografia;
private javax.swing.JMenuItem jmSair;
private javax.swing.JPopupMenu.Separator jsSeparador1;
private javax.swing.JPopupMenu.Separator jsSeparador2;
private javax.swing.JLabel labelConsAvancada;
private javax.swing.JLabel labelTitulo;
private javax.swing.JMenu menuAdm;
private javax.swing.JTable tableDoc;
private javax.swing.JTextArea textConsultaAvancada;
private javax.swing.JTextField tfBusca;
private javax.swing.JTextField tfTituloDoc;
// End of variables declaration

```

```

/**
 * @return the tableDoc
 */
public javax.swing.JTable getTableDoc() {
    return tableDoc;
}

/**
 * @return the tfTituloDoc
 */
public javax.swing.JTextField getTfTituloDoc() {
    return tfTituloDoc;
}

/**
 * @param tfTituloDoc the tfTituloDoc to set
 */
public void setTfTituloDoc(javax.swing.JTextField tfTituloDoc) {
    this.tfTituloDoc = tfTituloDoc;
}

void conectar(DB db, String user, String pwd) {

    try {
        mongo = new Mongo("localhost", 27017);
        db = mongo.getDB("repositorio");
        boolean aut = db.authenticate(user, pwd.toCharArray());
        myFS = new GridFS(db);
    } catch (UnknownHostException ex) {
        ex.printStackTrace();
    }
}

private BasicDBObject formataConsulta(String chave, String busca) {
    String exprReg = "^.*" + busca + ".*$";
    Pattern valor = Pattern.compile(exprReg, Pattern.CASE_INSENSITIVE);
    BasicDBObject query = new BasicDBObject("metadata." + chave, valor);
    return query;
}

private void consultaBanco(BasicDBObject query) {

    try {
        DBCursor cursor = myFS.getFileList(query);
        while (cursor.hasNext()) {
            BasicDBObject metadados = (BasicDBObject)
cursor.next().get("metadata");
            String mTitulo = (String) metadados.get("Titulo");
            String mAutor = (String) metadados.get("Autor");
            String mOrientador = (String) metadados.get("Orientador");
            String mAno = (String) metadados.get("Ano");
            Metadado m = new Metadado(mTitulo, mAutor, mOrientador, mAno);
            inserirTable(m);
        }
    } catch (Exception ex) {
        System.out.println("Sem resultado...");
    }
}
}

```

```

private void selecao() {
    int index = getTableDoc().getSelectedRow();
    if (index == -1) {
        return;
    }
    Metadado m = tableModelDoc.selecao(index);
    getTfTituloDoc().setText(m.getTitulo());
}

private void inserirTable(Metadado met) {
    if (met != null) {
        tableModelDoc.incluir(met);
    }
}

private void excluir() {
    int index = getTableDoc().getSelectedRow();
    if (index == -1) {
        JOptionPane.showConfirmDialog(null,
            " Nenhum registro selecionado! ", "Alerta!",
            JOptionPane.CLOSED_OPTION);
        return;
    }
    Metadado m = tableModelDoc.selecao(index);
    String titulo = m.getTitulo();
    BasicDBObject id = new BasicDBObject("metadata.Titulo", titulo);

    int selecionaOpcao = JOptionPane.showConfirmDialog(null,
        " Deseja excluir a seguinte monografia: " + titulo + "?",
        "Confirmação de exclusão",
        JOptionPane.OK_CANCEL_OPTION);
    if (selecionaOpcao == JOptionPane.OK_OPTION) {

        myFS.remove(id);

        tableModelDoc.excluir(index);
        JOptionPane.showConfirmDialog(null,
            " Exclusão efetuada com sucesso ", "Aviso!",
            JOptionPane.CLOSED_OPTION);
    }
}

private void limparTela() {

    int x = tableModelDoc.getRowCount();
    for (int i = 0; i < x; i++) {
        tableModelDoc.limparTela();
    }
    tfTituloDoc.setText("");
}

// metodo responsavel por editar os metadados
private void editar() {
    int index = getTableDoc().getSelectedRow();
    if (index == -1) {
        JOptionPane.showConfirmDialog(null,
            " Nenhum registro selecionado! ", "Alerta!",

```

```

        JOptionPane.CLOSED_OPTION);
    return;
}
Metadado m = tableModelDoc.selecao(index);
BasicDBObject query = new BasicDBObject("metadado.Titulo",
m.getTitulo());
GridFSDBFile fileout = myFS.findOne(query);
ObjectId id = (ObjectId) fileout.getId();

BasicDBObject metadados = (BasicDBObject) fileout.get("metadado");
String mTitulo = (String) metadados.get("Titulo");
String mAutor = (String) metadados.get("Autor");
String mOrientador = (String) metadados.get("Orientador");
String mTipo = (String) metadados.get("Tipo");
String mAno = (String) metadados.get("Ano");
String mSemestre = (String) metadados.get("Semestre");
String mResumo = (String) metadados.getString("Resumo");

EditarDocumento telaEditarDoc = new EditarDocumento(myFS, id,
mTitulo, mAutor, mOrientador, mTipo, mAno, mSemestre, mResumo);

telaEditarDoc.setVisible(true);
}

// metodo para detalhar um registro
private void detalhar() {
    int index = getTableDoc().getSelectedRow();
    if (index == -1) {
        JOptionPane.showConfirmDialog(null,
            " Nenhum registro selecionado! ", "Alerta!",
            JOptionPane.CLOSED_OPTION);
        return;
    }
    Metadado m = tableModelDoc.selecao(index);
    BasicDBObject query = new BasicDBObject("metadado.Titulo",
m.getTitulo());
    GridFSDBFile fileout = myFS.findOne(query);
    if (fileout == null) {
        JOptionPane.showConfirmDialog(null,
            " 0 registro selecionado foi alterado. É preciso refazer a
consulta ", "Alerta!",
            JOptionPane.CLOSED_OPTION);
        return;
    }
    ObjectId id = (ObjectId) fileout.getId();
    BasicDBObject metadados = (BasicDBObject) fileout.get("metadado");
    String mTitulo = (String) metadados.get("Titulo");
    String mAutor = (String) metadados.get("Autor");
    String mOrientador = (String) metadados.get("Orientador");
    String mTipo = (String) metadados.get("Tipo");
    String mAno = (String) metadados.get("Ano");
    String mSemestre = (String) metadados.get("Semestre");
    String mResumo = (String) metadados.getString("Resumo");

    VisualizarDetalhes telaVisualizarDoc = new VisualizarDetalhes(id,
mTitulo, mAutor, mOrientador, mTipo, mAno, mSemestre, mResumo);
    telaVisualizarDoc.setVisible(true);
}
}
}

```

A.3 NovoDocumento.java

```

package aplicacao;

import com.mongodb.BasicDBObject;
import com.mongodb.DBCursor;
import com.mongodb.gridfs.GridFS;
import com.mongodb.gridfs.GridFSInputFile;
import java.io.*;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.swing.JOptionPane;

/**
 *
 * @author ivan
 */
class FiltroArquivosPDF extends javax.swing.filechooser.FileFilter {

    @Override
    public boolean accept(File file) {
        // Allow only directories, or files with ".xml" extension
        return file.isDirectory() || file.getAbsolutePath().endsWith(".pdf");
    }

    @Override
    public String getDescription() {
        return "Arquivos PDF (*.pdf)";
    }
}

public class NovoDocumento extends javax.swing.JFrame {

    GridFS myFS;

    /**
     * Creates new form NovoDocumento
     */
    public NovoDocumento(GridFS myFS) {
        this.myFS = myFS;
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the
    form.
     * WARNING: Do NOT modify this code. The content of this method is
    always
     * regenerated by the Form Editor.
     */
}

```



```

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">

// </editor-fold>

private void buttonInserirActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:
    inserir();
}

private void buttonLimparActionPerformed(java.awt.event.ActionEvent evt)
{
    // TODO add your handling code here:
    limpar();
}

private void botaoBuscarPDFActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:
    fileChooser.setCurrentDirectory(new java.io.File("C:\\"));
    fileChooser.getSelectedFile();
    fileChooser.setFileFilter(new FiltroArquivosPDF());
    fileChooser.setVisible(true);
    jFrame1.pack();
    jFrame1.setVisible(true);
}

// Acao do botao Abrir
private void fileChooserActionPerformed(java.awt.event.ActionEvent evt)
{
    switch (evt.getActionCommand()) {
        case "ApproveSelection":
            File f = fileChooser.getSelectedFile();
            getTfArquivoPdf().setText(f.getAbsolutePath());
            jFrame1.dispose();
            break;
        case "CancelSelection":
            jFrame1.dispose();
            break;
    }
}

private void buttonCancelarActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:
    dispose();
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {

    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting
code (optional) ">

```

```

//</editor-fold>
/*
 * Create and display the form
 */
java.awt.EventQueue.invokeLater(new Runnable() {

    public void run() {
        new NovoDocumento(null).setVisible(true);
    }
});
}

// Variables declaration - do not modify
private javax.swing.JToggleButton botaoBuscarPDF;
private javax.swing.JButton buttonCancelar;
private javax.swing.JButton buttonInserir1;
private javax.swing.JButton buttonLimpar;
private javax.swing.JComboBox comboAno;
private javax.swing.JComboBox comboSemestre;
private javax.swing.JComboBox comboTipo;
private javax.swing.JFileChooser fileChooser;
private javax.swing.JFrame jFrame1;
private javax.swing.JPanel jPanel1;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JLabel labelAno;
private javax.swing.JLabel labelArquivo;
private javax.swing.JLabel labelAutor;
private javax.swing.JLabel labelOrientador;
private javax.swing.JLabel labelResumo;
private javax.swing.JLabel labelSemestre;
private javax.swing.JLabel labelTipo;
private javax.swing.JLabel labelTitulo;
private javax.swing.JTextArea textAreaResumo;
private javax.swing.JTextField tfArquivoPdf;
private javax.swing.JTextField tfAutor;
private javax.swing.JTextField tfOrientador;
private javax.swing.JTextField tfTitulo;
// End of variables declaration

/**
 * @return the tfTitulo
 */
public javax.swing.JTextField gettfTitulo() {
    return tfTitulo;
}

/**
 * @return the tfAutor
 */
public javax.swing.JTextField gettfAutor() {
    return tfAutor;
}

/**
 * @return the tfOrientador
 */
public javax.swing.JTextField gettfOrientador() {
    return tfOrientador;
}

```

```

/**
 * @return the labelTitulo
 */
public javax.swing.JLabel getlabelTitulo() {
    return labelTitulo;
}

/**
 * @return the comboAno
 */
public javax.swing.JComboBox getComboAno() {
    return comboAno;
}

/**
 * @return the comboSemestre
 */
public javax.swing.JComboBox getComboSemestre() {
    return comboSemestre;
}

/**
 * @return the comboTipo
 */
public javax.swing.JComboBox getComboTipo() {
    return comboTipo;
}

/**
 * @return the tfArquivoPdf
 */
public javax.swing.JTextField getTfArquivoPdf() {
    return tfArquivoPdf;
}

/**
 * @return the textAreaResumo
 */
public javax.swing.JTextArea getTextAreaResumo() {
    return textAreaResumo;
}

/**
 * @return the labelNota
 */
public javax.swing.JLabel getLabelNota() {
    return labelAno;
}

private void inserir() {
    String textTitulo = labelTitulo.getText();
    String titulo = tfTitulo.getText();
    String textAutor = labelAutor.getText();
    String autor = tfAutor.getText();
    String textOrientador = labelOrientador.getText();
    String orientador = tfOrientador.getText();
    String tipoDoc = labelTipo.getText();
    String tipo = (String) getComboTipo().getSelectedItem();
    String textAno = labelAno.getText();
}

```

```

String ano = (String) getComboAno().getSelectedItem();
String textSemestre = labelSemestre.getText();
String semestre = (String) getComboSemestre().getSelectedItem();
String textResumo = labelResumo.getText();
String resumo = getTextAreaResumo().getText();

BasicDBObject metadados = new BasicDBObject();
metadados.put(textTitulo, titulo);
metadados.put(textAutor, autor);
metadados.put(textOrientador, orientador);
metadados.put(tipoDoc, tipo);
metadados.put(textAno, ano);
metadados.put(textSemestre, semestre);
metadados.put(textResumo, resumo);

String pathArquivoPdf = getTfArquivoPdf().getText();

if ("".equals(titulo) || "".equals(autor) || "".equals(orientador) ||
"".equals(tipo)
    || "".equals(ano) || "".equals(semestre) ||
"".equals(pathArquivoPdf) || "".equals(resumo)) {
    JOptionPane.showConfirmDialog(null,
        " Todos os campos devem estar preenchidos! ", "Alerta!",
        JOptionPane.CLOSED_OPTION);
} else {
    insereDocumento(metadados, pathArquivoPdf);
}
}

// Metodo para inserir o documento na Base de Dados
private void insereDocumento(BasicDBObject metadados, String
pathArquivoPdf) {

    File file = new File(pathArquivoPdf);
    String md5 = null;
    try {
        md5 = geraHash(file);
    } catch (NoSuchAlgorithmException | FileNotFoundException e1) {
    }

    boolean existeArquivo = verificaExistePdf(md5, myFS);

    if (!existeArquivo) {

        try {
            GridFSInputFile inputFile = myFS.createFile(file);
            inputFile.setMetaData(metadados);
            inputFile.save();

            JOptionPane.showConfirmDialog(null,
                " Nova monografia guardada com sucesso! ", null,
                JOptionPane.CLOSED_OPTION);
            dispose();
        } catch (IOException ex) {
        }
    } else {
        JOptionPane.showConfirmDialog(null,
            " O arquivo PDF já existe na base de dados! ", null,
            JOptionPane.CLOSED_OPTION);
    }
}

```

```

    }
}

// Gera a lista dos anos no combobox
public String[] buscarAnos() {
    String dados[] = null;
    Date hoje = new Date();
    String formato = "yyyy";
    SimpleDateFormat formatter = new SimpleDateFormat(formato);
    int anoatual = Integer.parseInt(formatter.format(hoje));
    dados = new String[anoatual - 1898];
    dados[0] = "";

    for (int i = 1; anoatual >= 1900; i++) {
        dados[i] = String.valueOf(anoatual);
        anoatual--;
    }

    return dados;
}

// Verifica se existe a Hash md5 no gridfs
private boolean verificaExistePdf(String md5, GridFS myFS) {
    boolean existe = false;
    BasicDBObject query = new BasicDBObject("md5", md5);
    DBCursor cursor = myFS.getFileList(query);
    if (cursor.hasNext()) {
        return true;
    } else {
        return false;
    }
}

//Gera a hash md5 do arquivo a ser inserido
public String geraHash (File f) throws NoSuchAlgorithmException,
FileNotFoundException {
    MessageDigest digest = MessageDigest.getInstance("MD5");
    InputStream is = new FileInputStream(f);
    byte[] buffer = new byte[8192];
    int read = 0;
    String output = null;
    try {
        while ((read = is.read(buffer)) > 0) {
            digest.update(buffer, 0, read);
        }
        byte[] md5sum = digest.digest();
        BigInteger bigInt = new BigInteger(1, md5sum);
        output = bigInt.toString(16);
    } catch (IOException e) {
        throw new RuntimeException("Não foi possível processar o
arquivo.", e);
    } finally {
        try {
            is.close();
        } catch (IOException e) {
            throw new RuntimeException("Não foi possível fechar o arquivo",
e);
        }
    }
}
}

```

```

        return output;
    }

    // Limpa todos os campos
    private void limpar() {
        gettfTitulo().setText("");
        gettfAutor().setText("");
        getTfOrientador().setText("");
        getComboAno().setSelectedIndex(-1);
        getComboSemestre().setSelectedIndex(0);
        getTextAreaResumo().setText("");
    }
}

```

A.4 EditarDocumento.java

```

package aplicacao;
import com.mongodb.BasicDBObject;
import com.mongodb.gridfs.GridFS;
import com.mongodb.gridfs.GridFSDBFile;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.swing.JOptionPane;
import org.bson.types.ObjectId;

public class EditarDocumento extends javax.swing.JFrame {

    GridFS myFS;
    ObjectId id;
    String titulo;
    String autor;
    String orientador;
    String tipo;
    String ano;
    String semestre;
    String resumo;

    public EditarDocumento(GridFS myFS, ObjectId id, String titulo, String
    autor, String orientador,
        String tipo, String ano, String semestre, String resumo) {

        this.myFS = myFS;
        this.id = id;
        this.titulo = titulo;
        this.autor = autor;
        this.orientador = orientador;
        this.tipo = tipo;
        this.ano = ano;
        this.semestre = semestre;
    }
}

```

```

        this.resumo = resumo;
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the
form.
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">

} // </editor-fold>

private void buttonSalvarActionPerformed(java.awt.event.ActionEvent evt)
{
    // TODO add your handling code here
    editar();
}

private void buttonCancelarActionPerformed(java.awt.event.ActionEvent
evt) {
    // TODO add your handling code here:
    dispose();
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {

    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting
code (optional) ">
    //</editor-fold>

    /*
     * Create and display the form
     */
    java.awt.EventQueue.invokeLater(new Runnable() {

        public void run() {
            new EditorDocumento(null, null, null, null, null, null, null,
null, null).setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton buttonCancelar;
private javax.swing.JButton buttonSalvar;
private javax.swing.JComboBox comboAno;
private javax.swing.JComboBox comboSemestre;
private javax.swing.JComboBox comboTipo;
private javax.swing.JPanel jPanel1;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JLabel labelAno;
private javax.swing.JLabel labelAutor;

```

```

private javax.swing.JLabel labelOrientador;
private javax.swing.JLabel labelResumo;
private javax.swing.JLabel labelSemestre;
private javax.swing.JLabel labelTipo;
private javax.swing.JLabel labelTitulo;
private javax.swing.JTextArea textAreaResumo;
private javax.swing.JTextField tfAutor;
private javax.swing.JTextField tfOrientador;
private javax.swing.JTextField tfTitulo;
// End of variables declaration

// Captura os metadados e chama o metodo para guardar na Base de Dados
private void editar() {

    String textTitulo = labelTitulo.getText();
    titulo = tfTitulo.getText();
    String textAutor = labelAutor.getText();
    autor = tfAutor.getText();
    String textOrientador = labelOrientador.getText();
    orientador = tfOrientador.getText();
    String tipoDoc = labelTipo.getText();
    tipo = (String) comboTipo.getSelectedItem();
    String textAno = labelAno.getText();
    ano = (String) comboAno.getSelectedItem();
    String textSemestre = labelSemestre.getText();
    semestre = (String) comboSemestre.getSelectedItem();
    String textResumo = labelResumo.getText();
    resumo = textAreaResumo.getText();
    //resumo = getTextAreaResumo().getText();

    BasicDBObject metadados = new BasicDBObject();
    metadados.put(textAutor, autor);
    metadados.put(textTitulo, titulo);
    metadados.put(textOrientador, orientador);
    metadados.put(tipoDoc, tipo);
    metadados.put(textAno, ano);
    metadados.put(textSemestre, semestre);
    metadados.put(textResumo, resumo);

    if ("".equals(titulo) || "".equals(autor) || "".equals(orientador) ||
"".equals(tipo)
        || "".equals(ano) || "".equals(semestre) || "".equals(resumo)) {
        JOptionPane.showConfirmDialog(null,
            " Todos os campos devem estar preenchidos! ", "Alerta!",
            JOptionPane.CLOSED_OPTION);
    } else {
        editarMetadados(id, metadados);
    }
}

// Metodo para gerar os anos no combobox
public String[] buscarAnos() {
    String dados[] = null;
    Date hoje = new Date();
    String formato = "yyyy";
    SimpleDateFormat formatter = new SimpleDateFormat(formato);
    int anoatual = Integer.parseInt(formatter.format(hoje));
    dados = new String[anoatual - 1898];
}

```



```

    dados[0] = "";

    for (int i = 1; anoatual >= 1900; i++) {
        dados[i] = String.valueOf(anoatual);
        anoatual--;
    }
    return dados;
}

// Editar os metadados na Base de Dados
private void editarMetadados (ObjectId id, BasicDBObject metadados) {

    GridFSDBFile file = myFS.findOne(id);
    file.setMetaData(metadados);
    file.save();
    dispose();
}
}

```

A.5 VisualizarDetalhes.java

```

package aplicacao;
import org.bson.types.ObjectId;

public class VisualizarDetalhes extends javax.swing.JFrame {

    ObjectId id;
    String titulo;
    String autor;
    String orientador;
    String tipo;
    String ano;
    String semestre;
    String resumo;

    public VisualizarDetalhes (ObjectId id, String titulo, String autor,
        String orientador, String tipo, String ano, String semestre,
        String resumo) {

        this.id = id;
        this.titulo = titulo;
        this.autor = autor;
        this.orientador = orientador;
        this.tipo = tipo;
        this.ano = ano;
        this.semestre = semestre;
        this.resumo = resumo;

        initComponents();
    }
}

```

```

/**
 * This method is called from within the constructor to initialize the
 form.
 * WARNING: Do NOT modify this code. The content of this method is
 always
 * regenerated by the Form Editor.
 */

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">

// </editor-fold>

private void buttonFecharActionPerformed(java.awt.event.ActionEvent evt)
{
    // TODO add your handling code here:
    dispose();
}

public static void main(String args[]) {

    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting
code (optional) ">
    //</editor-fold>

    /**
     * Create and display the form
     */
    java.awt.EventQueue.invokeLater(new Runnable() {

        public void run() {
            new VisualizarDetalhes(null, null, null, null, null,
                null, null, null).setVisible(true);
        }
    });
}
// Variables declaration - do not modify
private javax.swing.JButton buttonFechar;
private javax.swing.JPanel jPanel1;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JLabel labelAno;
private javax.swing.JLabel labelAutor;
private javax.swing.JLabel labelOrientador;
private javax.swing.JLabel labelResumo;
private javax.swing.JLabel labelSemestre;
private javax.swing.JLabel labelTipo;
private javax.swing.JLabel labelTitulo;
private javax.swing.JTextArea textAreaResumo;
private javax.swing.JTextField tfAno;
private javax.swing.JTextField tfAutor;
private javax.swing.JTextField tfOrientador;
private javax.swing.JTextField tfSemestre;
private javax.swing.JTextField tfTipo;
private javax.swing.JTextField tfTitulo;
// End of variables declaration
}

```

A.6 TableModelDocjava

```

package aplicacao;
import java.util.ArrayList;
import javax.swing.table.AbstractTableModel;

public class TableModelDoc extends AbstractTableModel {
    private static final String[] columnNames = {"Titulo", "Autor",
"Orientador", "Ano"};
    private ArrayList<Metadado> docs;

    public TableModelDoc() {
        docs = new ArrayList<>();
    }
    public void incluir(Metadado l) {
        docs.add(l);
        fireTableRowsInserted(docs.size() - 1, docs.size() - 1);
    }
    public void excluir(int index) {
        docs.remove(index);
        fireTableRowsDeleted(1, getRowCount());
    }
    public void limparTela() {
        docs.remove(0);
        fireTableRowsDeleted(1, getRowCount());
    }
    public Metadado selecao(int index) {
        return docs.get(index);
    }

    @Override
    public int getRowCount() {
        return docs.size();
    }
    @Override
    public String getColumnName(int columnIndex) {
        return columnNames[columnIndex];
    }
    @Override
    public int getColumnCount() {
        return columnNames.length;
    }
    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        switch (columnIndex) {
            case 0:
                return docs.get(rowIndex).getTitulo();
            case 1:
                return docs.get(rowIndex).getAutor();
            case 2:
                return docs.get(rowIndex).getOrientador();
            case 3:
                return docs.get(rowIndex).getAno();
        }
        return null;
    }
}

```

A.7 Metadado.java

```
package aplicacao;

public class Metadado {

    private String titulo;
    private String autor;
    private String orientador;
    private String ano;

    public Metadado() {
    }

    public Metadado(String titulo, String autor, String orientador, String
ano) {
        this.titulo = titulo;
        this.autor = autor;
        this.orientador = orientador;
        this.ano = ano;
    }

    public String getTitulo() {
        return titulo;
    }
    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }
    public String getAutor() {
        return autor;
    }
    public void setAutor(String autor) {
        this.autor = autor;
    }
    public String getOrientador() {
        return orientador;
    }
    public void setOrientador(String orientador) {
        this.orientador = orientador;
    }
    public String getAno() {
        return ano;
    }
    public void setAno(String ano) {
        this.ano = ano;
    }
}
```