

FACULDADE DE BALSAS
CURSO DE SISTEMAS DE INFORMAÇÃO

**DESENVOLVIMENTO DE UMA FERRAMENTA PARA INDEXAÇÃO
E BUSCA DE DOCUMENTOS EMPRESARIAIS**

Igor Ribeiro Santos

BALSAS – MA
2013

FACULDADE DE BALSAS
CURSO DE SISTEMAS DE INFORMAÇÃO

**DESENVOLVIMENTO DE UMA FERRAMENTA PARA INDEXAÇÃO
E BUSCA DE DOCUMENTOS EMPRESARIAIS**

Igor Ribeiro Santos

Trabalho de conclusão de curso para Curso de
Sistemas de Informação como requisito para a
obtenção do grau de Bacharel em Sistemas de
Informação.

Orientador: Jefferson Fontinele da Silva

Coordenador do Curso de Sistemas de Informação: Junior Marcos Bandeira

Balsas - MA
2013

Dedico este trabalho a meus pais, Manoel e Maria do Carmo, e em memória de minha querida avó Raimunda Ribeiro, exemplos de perseverança e maturidade que, com o passar dos anos não se endureceram, mas adquiriram a virtude da mudança à medida que o tempo passa. Isso realmente é deixar Deus trabalhar dentro de si. Eu amo vocês.

AGRADECIMENTOS

Agradeço primeiramente a Deus, ao professor Jefferson Fontinele pela paciência na orientação e incentivo que tornaram possível a conclusão deste trabalho, aos meus pais, irmãos, a todos os professores que fizeram parte da minha vida acadêmica.

Por fim aos meus amigos e colegas pelo incentivo e apoio constantes, em especial aos meus amigos Edson Maia e Fernando Matos, pela parceria nestes árduos e inesquecíveis anos de faculdade.

RESUMO

As empresas cada vez mais vêm produzindo informações sobre todos os processos realizados dentro dela e documentando-as, assim tem-se que pensar em ferramentas que auxiliem e facilitem a busca e organização destes documentos. Assumindo essa hipótese este trabalho apresenta o desenvolvimento de uma ferramenta de busca e indexação em banco de dados de documentos, a fim de tornar mais eficiente a gestão da documentação nas empresas. Para desenvolvimento desta ferramenta foram utilizados, o framework GRAILS, o GGTS como ambiente de desenvolvimento, o Elasticsearch como servidor de busca e indexação de documentos, o banco de dados não relacional (NOSQL) e orientado à documentos MongoDB, e o plugin do Apache POI para leitura dos documentos.

Palavras-chave: Documentos; Indexação; Busca.

ABSTRACT

Companies increasingly are producing information about all processes performed within it and documenting it, so one has to think of tools that assist and facilitate search and organization of these documents. Assuming this hypothesis this work presents the development of a tool for search and indexing database documents in order to make more efficient document management in companies. For development of this tool were used, the Grails framework, the GGTS as environment development, as elasticsearch server search and indexing documents, the database non-relational (NoSQL) and document-oriented MongoDB, and plugin Apache POI for reading documents.

Keywords: Documents, Indexing, Search.

LISTA DE FIGURAS

Figura 1: Objeto.....	26
Figura 2: Matriz.....	26
Figura 3: Valor.....	27
Figura 4: String.....	27
Figura 5: Número.....	27
Figura 6: Cachos ElasticSearch	28
Figura 7: Ambiente GGTS	32
Figura 8: Arquitetura do Sistema de Busca e Indexação de documentos	34
Figura 9: Listagem de Documentos	41
Figura 10: Criação de novo Documento	42
Figura 11: Tela Show Documento.....	42
Figura 12: Tela de Busca.....	43

LISTA DE TABELAS

Tabela 1: Análise Comparativa Modelo Relacional X NOSQL	19
Tabela 2: Requisitos do Sistema de Busca e Indexação de documentos	35

GLOSSÁRIO

API - é a sigla de *Application Programming Interface* ou, em português, Interface de Programação de Aplicativo, esta interface é o conjunto de padrões de programação que permite a construção de aplicativos e a sua utilização de maneira não tão evidente para os usuários.

Cluster - (*clustering*) é o nome dado a um sistema que relaciona dois ou mais computadores para que estes trabalhem de maneira conjunta no intuito de processar uma tarefa. Estas máquinas dividem entre si as atividades de processamento e executam este trabalho de maneira simultânea.

HTTP - *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto) é um protocolo de comunicação (na camada de aplicação segundo o Modelo OSI) utilizado para sistemas de informação de hipermedia distribuídos e colaborativos.

JavaScript - é uma linguagem de programação interpretada . Foi originalmente implementada como parte dos navegadores *web* para que *scripts* pudessem ser executados do lado do cliente e interagissem com o usuário sem a necessidade deste script passar pelo servidor, controlando o navegador, realizando comunicação assíncrona e alterando o conteúdo do documento exibido.

MapReduce - é considerado um novo modelo computacional distribuído, inspirado pelas funções *map* e *reduce* usadas comumente em programação funcional. O processo de mapear a requisição do originador para o *data source* é chamado de '*Map*', e o processo de agregação do resultado em um resultado consolidado é chamado de '*Reduce*'.

MVC - é um conceito (paradigma) de desenvolvimento e design que tenta separar uma aplicação em três partes distintas. Uma parte, a *Model*, está relacionada ao trabalho atual que a aplicação administrativa, outra parte, a *View*, está relacionada a exibir os dados ou informações dessa uma aplicação e a terceira parte, *Controller*, em coordenar os dois anteriores exibindo a interface correta ou executando algum trabalho que a aplicação precisa completar.

MVCC (*Multiversion Concurrency Control*) - tem como objetivo auxiliar na implementação do padrão ACID (atomicidade, consistência, isolamento e durabilidade), onde cada transação permanece com a imagem do estado do banco de dados no início da transação, ou seja, podem existir várias versões do banco ao mesmo tempo, um para cada transação.

SUMÁRIO

1 - INTRODUÇÃO.....	13
1.1 – Justificativa.....	13
1.2 – Objetivos.....	14
1.2.1 – Objetivo Geral	14
1.2.2 – Objetivos Específicos.....	14
1.3 – Metodologia.....	14
1.4 – Organização da Monografia	15
2 – REVISÃO BIBLIOGRÁFICA	16
2.1 – Banco de Dados NOSQL	16
2.1.1 – Classificação de Banco de Dados NOSQL.....	17
2.1.2 – NOSQL X Bancos de Dados Relacionais.....	18
2.1.3 – Características de alguns Bancos de Dados NOSQL.....	20
2.2 – JSON (JavaScript Object Notation).....	25
2.3 – Elasticsearch	28
2.3.1 – Características do Elasticsearch	28
2.4 – Sphinx (Esfinge)	29
2.4.1 – Características da Sphinx	29
2.5– Groovy.....	30
2.6– GRAILS.....	31
2.7 – GGTS (Groovy/Grails Tool Suite)	31
2.8 – Exemplo aplicação GRAILS construída com GGTS Groovy	32
2.8.1 – Criação do Domínio.....	32
2.8.2 – Criação do Controlador	33
2.8.3 – Adicionando o Plugin Elasticsearch à aplicação.....	33
2.9 – Apache POI.....	34
2.9.1 – Apache POI - HWPf - Java API para lidar com arquivos do Microsoft Word	34
2.10 – Arquitetura do Sistema de Busca e Indexação de documentos	34
2.11 – Requisitos do Sistema de Busca e Indexação de documentos	35
3 – DOCINDEX	36
3.1– Configuração do Banco de Dados MongoDB.....	36
3.2 – Instalação do Elasticsearch no Windows.....	36
3.3 – Plugins instalados	36

3.4 – Domínios	37
3.4.1– Documento.groovy	37
3.4.1 – Paragrafo.groovy	38
3.5 – Controladores	38
3.5.1 – Action Save	38
3.5.2 – Action Delete	39
3.5.3 – Action Search	39
3.5.4 – Action Download	40
3.5.5 – Action UploadForm	40
3.6 – Views	41
3.7 – Telas do Sistema DocIndex	41
3.8 – Dificuldades Encontradas.....	43
4 – CONCLUSÃO E ATIVIDADES FUTURAS	44
5 – REFERÊNCIAS.....	45

1 INTRODUCAO

A indexação de documentos consiste na identificação de seus traços descritivos e em seguida extrair seus elementos descritores indicadores do seu conteúdo visando sua recuperação posterior, sendo que estes descritores vão apenas representar os indicadores do conteúdo do documento e não a sua representação como um todo, pois esta só pode ser feita por ele próprio. (GARDIN, 1974).

Devido a grande quantidade de dados e informações produzidos todos os dias nas organizações faz-se necessário criar mecanismos e ferramentas que facilitem e agilizem a busca posterior em bancos de dados de documentos. Indexar documentos, ou seja, criar índices para eles é uma forma de organização e que por consequência, acaba com a dificuldade de localizá-los depois de armazenados.

Uma das formas de armazenar documentos é a através da utilização de banco de dados NOSQL, como são apresentados pelo *Yahoo*, *Twitter*, *Facebook*, *LinkedIn* e *Google* (BRITO,2010). O termo *NOSQL* surgiu em 1998, a partir de uma solução de banco de dados que não oferecia uma interface *SQL*, mas esse sistema ainda era baseado na arquitetura relacional. Posteriormente, o termo passou a representar soluções que promoviam uma alternativa ao modelo relacional, tornando-se uma abreviação de *Not Only SQL* (não apenas *SQL*), sendo utilizado principalmente em casos em que o modelo relacional não apresentava performance adequada.

Os bancos de dados *NOSQL* surgiram como uma solução para a questão da escalabilidade no armazenamento e processamento de grandes volumes de dados, este tipo de banco de dados apresenta as seguintes características: não-relacional, distribuído, escalável horizontalmente, ausência de esquema ou esquema flexível, suporte à replicação nativa e acesso via *APIs* simples. (BRITO,2010).

Essa monografia está inserida na área de desenvolvimento de *software*, de forma mais específica o desenvolvimento de uma ferramenta *Web* de Busca e Indexação em banco de dados de documentos (*NOSQL*), que possui esquema livre, distribuído e orientado a documentos.

1.1 JUSTIFICATIVA

Um executivo pode perder quatro semanas em um ano procurando informações e documentos; gasta-se 250 dólares para recriar um documento perdido; guarda-se 3 vezes mais

documentos do que o necessário; em cada 20 documentos, um é perdido; faz-se 19 cópias de um documento. [COOPERS & LYBRAND]

As organizações vêm cada vez mais produzindo informações e documentando-as, isso pode acarretar em um grande acúmulo de documentos e até mesmo perda dos mesmos, cujo conteúdo é de importância. As estatísticas citadas apresentadas anteriormente demonstram a dificuldade que as empresas enfrentam em relação aos documentos produzidos diariamente, a partir disso faz-se necessário o desenvolvimento de uma ferramenta que faça indexação e busca de forma rápida e eficiente qualquer documento relativo à empresa.

1.2 OBJETIVOS

1.2.1 Objetivo geral:

Desenvolver um sistema de busca e indexação de documentos escalável e distribuído, que deve atender aos requisitos de armazenamento de documentos de uma empresa de médio porte.

1.2.2 Objetivos específicos:

Verificar a utilização prática de banco de dados orientado a documentos.

Identificar sistemas distribuídos de indexação de documentos.

Analisar os requisitos necessários para sistemas de busca de documentos.

1.3 METODOLOGIA

A metodologia que foi utilizada consistiu na identificação de requisitos que atendam o sistema de indexação e busca de documentos, através de questionários, pesquisa e estudos bibliográficos em livros, revistas, periódicos e publicações diversas.

Realização de testes comparativos, modo pelo qual foram definidas as ferramentas. Utilização da prototipagem em ambiente simulado, como metodologia de desenvolvimento de software.

Realização de testes em plataformas clientes diferentes para o serviço de busca de documentos.

O desenvolvimento da ferramenta foi dividida em fases onde, em cada uma dessas fases foi desenvolvido uma funcionalidade, quando todas ficaram prontas, integrou-se todas formando o sistema de busca e indexação de documentos DocIndex.

1.4 ORGANIZAÇÃO DA MONOGRAFIA

Este trabalho está organizado da seguinte forma: Capítulo 1 Introdução, onde de encontra toda ideia do projeto; Capítulo 2 Revisão Bibliográfica, que contém todo o referencial bibliográfico necessário para o desenvolvimento do projeto; Capítulo 3 DocIndex, onde se encontra as práticas utilizadas para o desenvolvimento da ferramenta; Capítulo 4 Conclusão, que apresenta os resultados do projeto; Um Anexo, contendo uma Query Elasticsearch; e um Apêndice, contendo o código das *Views* da aplicação.

2. REVISÃO BIBLIOGRÁFICA

Neste capítulo serão abordados os conceitos necessários para desenvolvimento do trabalho de acordo com artigos de eventos relacionados ao tema em questão.

2.1 Banco de Dados NOSQL

Os bancos de dados *NOSQL* surgiram como uma solução para a questão da escalabilidade no armazenamento e processamento de grandes volumes de dados. No início, grandes empresas enfrentando esse tipo de problema criaram suas próprias soluções, e publicaram alguns artigos propondo diversas soluções ligadas ao gerenciamento de dados distribuído em larga escala, mas sem usar o nome *NOSQL*. O nome só surgiu alguns anos depois, em 2009, quando algumas novas empresas da *Web 2.0* e a comunidade de software livre e código aberto começaram a desenvolver novas opções de bancos de dados. (BRITO, 2010)

O objetivo dos projetistas de banco de dados de organizações de grande porte passou a ser desenvolver uma nova forma de armazenamento na qual pudessem estar livres de certas estruturas e regras do Modelo Relacional. Assim, foram surgindo soluções que pareciam voltar no tempo, retornando ao simples sistemas de gerenciamento de arquivos, se, por um lado, essas soluções perdiam todo o arcabouço de regras de consistência presentes no Modelo Relacional, por outro, poderiam ganhar em desempenho, flexibilizando os sistemas de banco de dados para as características particulares de cada organização.

O propósito, portanto, das soluções *NOSQL* não é substituir o Modelo Relacional como um todo, mas apenas em casos nos quais seja necessária uma maior flexibilidade da estruturação do banco.

Uma das primeiras implementações de um sistema realmente não relacional surgiu em 2004 quando o *Google* lançou o *BigTable* (BRITO, 2010), um banco de dados proprietário de alta performance que tinha como objetivo promover maior escalabilidade e disponibilidade. A ideia central era justamente flexibilizar a forte estruturação utilizada pelo Modelo Relacional.

Outra implementação foi realizada em 2007, pela *Amazon*, com a apresentação do sistema *Dynamo* (BRITO, 2010). , o qual tinha como característica básica ser um banco de dados não-relacional, de alta disponibilidade, utilizado pelos *web services* da *Amazon*.

Em 2008, iniciou-se outro importante projeto. O Cassandra foi projetado para ser um sistema de banco de dados distribuído e de alta disponibilidade, desenvolvido pelo site *Facebook* para lidar com grandes volumes de dados. No início de 2010, o Cassandra desbancou o *MySQL* como banco de dados do *Twitter*, demonstrando sua importância cada vez mais crescente. (BRITO, 2010)

Na mesma época, começou o desenvolvimento do *Apache CouchDB*, um banco de dados livre e orientado a documentos, os quais armazenam os dados sem a necessidade de qualquer esquema pré-definido, e que utiliza o conceito de arquitetura *MapReduce*, especialmente projetada para suportar computação distribuída em larga escala. (BRITO, 2010) (COUCHDB, 2013)

Outra solução emergente, lançada em 2009, foi o *MongoDB*, um banco de dados orientado a documentos, escalável, de alta performance e livre de esquemas, com várias características semelhantes ao *CouchDB*. (BRITO, 2010)

2.1.1 Classificação de Banco de Dados NOSQL

Apesar de possuírem certas características em comum, tais como serem livres de esquema, promoverem alta disponibilidade e maior escalabilidade, os sistemas de bancos de dados *NOSQL* existentes possuem diversas singularidades. Quanto à distribuição de dados, certos sistemas promovem o particionamento e a replicação dos dados, enquanto outros deixam essa tarefa para o cliente. A maioria das soluções é distribuída, como é caso do *Amazon Dynamo*, *CouchDB*, *MongoDb*, *BigTable* e *Cassandra*. (BRITO, 2010)

Quanto ao modelo de dados, existem quatro categorias básicas: os sistemas baseados em armazenamento chave-valor, como é o caso do *Amazon Dynamo*; os sistemas orientados a documentos, entre os quais temos o *CouchDB* e o *MongoDB*; os sistemas orientados a coluna, que tem como exemplos o *Cassandra* e o *BigTable*; e os sistemas baseados em grafos, como são os casos do *Neo4j*, que armazena dados em nós conectados por relações dirigidas, digitados com propriedades de ambos, também conhecidos como um grafo de propriedade e do *InfoGrid*, que é um banco de dados baseado em grafos com componentes de software adicionais para o desenvolvimento web *RESTFUL* (*Representational State Transfer*), um estilo de projetar aplicativos web fracamente acoplados que contam com recursos nomeados, e não com mensagens. (BRITO, 2010)

Na primeira categoria, armazenamento chave-valor, existe uma coleção de chaves únicas e de valores, os quais são associados com as chaves. No segundo grupo, sistemas orientados a documentos, os documentos são as unidades básicas de armazenamento e estes não utilizam necessariamente qualquer tipo de estruturação pré-definida, como é o caso das tabelas do Modelo Relacional. Um documento do *CouchDB*, por exemplo, é um objeto que possui um identificador único e que consiste de certos campos. Esse documento segue o formato JSON (*JavaScript Object Notation*), padrão que visa uma fácil legibilidade em independência de linguagem. (BRITO, 2010)

Na terceira categoria, sistemas orientados a coluna, muda-se o paradigma de orientação a registros (ou tuplas) para orientação a atributos (ou colunas). O Cassandra possui colunas (tuplas que contêm nome, valor e *timestamp*), famílias de colunas (um repositório para colunas, análogo a uma tabela do Modelo Relacional) e super-colunas (compostas por *arrays* de colunas). (BRITO, 2010)

Na quarta categoria, sistemas baseados em grafos, os dados são armazenados em nós de um grafo cujas arestas representam o tipo de associação entre esses nós. (BRITO, 2010)

Quanto ao modo de armazenamento dos dados, tem-se os bancos que mantêm suas informações em memória realizando persistências ocasionais; aqueles que mantêm suas informações em disco, como são os casos do *CouchDb* e do *MongoDb*; e aqueles configuráveis, tais como *BigTable* e o *Cassandra*.

Após analisar as quatro categorias de banco de dados *NOSQL*, percebeu-se que a solução mais adequada para o desenvolvimento da ferramenta de busca e indexação de documentos são os bancos de dados orientados a documentos, como o *MongoDB*.

2.1.2 NOSQL X Bancos de Dados Relacionais

Quando se analisa a possibilidade de se optar por uma estratégia *NOSQL* ao invés de um Banco de Dados Relacional, é preciso levar em consideração algumas questões básicas, como observado na Tabela 1.

A questão do escalonamento é essencial porque é justamente nesse ponto em que os bancos *NOSQL* apresentam as principais vantagens em relação aos bancos de dados relacionais, basicamente pelo fato de terem sido criados para esse fim, enquanto os sistemas relacio-

nais possuem uma estruturação menos flexível e menos adaptada para cenários em que o escalonamento faz-se necessário.

Quanto à disponibilidade de dados, os bancos de dados *NOSQL* são superiores aos modelos relacionais, como exemplo disso, pode ser citado o caso do *Twitter*. Em 2008, enquanto ainda utilizava o *PostgreSQL* (modelo relacional), o site que funciona como rede social ficou 84 horas for do ar. Em 2009, após a utilização do *Cassandra* (*NOSQL*), esse tempo foi reduzido para 23 horas e 45 minutos. (BRITO, 2010)

Os principais benefícios da abordagem não relacional podem ser identificados como: maior disponibilidade, menor tempo de resposta para consultas, paralelismo de atualização de dados e maior grau de concorrência.

	Modelo Relacional	NOSQL
Escalonamento	Possível, mas complexo. Devido à natureza estruturada do modelo, a adição de forma dinâmica e transparente de novos nós <i>no grid</i> não é realizada de modo natural.	Uma das principais vantagens desse modelo. Por não possuir nenhum tipo de esquema pré-definido, o modelo possui maior flexibilidade o que favorece a inclusão transparente de outros elementos.
Consistência	Ponto mais forte do modelo relacional. As regras de consistência presentes propiciam um maior grau de rigor quanto à consistência das informações.	Realizada de modo eventual no modelo: só garante que, se nenhuma atualização for realizada sobre o item de dados, todos os acessos a esse item devolverão o último valor atualizado.
Disponibilidade	Dada a dificuldade de se conseguir trabalhar de forma eficiente com a distribuição dos dados, esse modelo pode não suportar a demanda muito grande de informações do banco.	Outro fator fundamental do sucesso desse modelo. O alto grau de distribuição dos dados propicia que um maior número de solicitações aos dados seja atendida por parte do sistema e que o sistema fique menos tempo não disponível.

Tabela 1: Análise Comparativa Modelo Relacional X NOSQL (BRITO, 2010)

2.1.3 Características de alguns Bancos de Dados NOSQL

Neo4j

Neo4j é um banco de dados orientado a grafos de código aberto apoiado por Neo Tecnologia, armazena dados em nós conectados por relações dirigidas, digitados com propriedades de ambos, também conhecidos como um grafo de propriedade. (NEO4J, 2013)

Principais características:

- Utiliza um modelo de grafo para representação de dados.
- Transações ACID completas.
- Massivamente escalável, até vários bilhões de nós, relações, propriedades.
- Altamente disponível, quando distribuídos em várias máquinas.
- Expressiva, com uma poderosa linguagem de consulta.
- Rápido, com um quadro de passagem poderoso para consultas de alta velocidade.
- Acessível por uma interface *REST* conveniente ou um objeto orientado a *API Java*.

MongoDB

MongoDB é um banco de dados de documentos que fornece alto desempenho, alta disponibilidade e escalabilidade. (THE MONGODB 2.4 MANUAL, 2013)

Principais características:

- Banco de dados de documentos
- Alto desempenho
- Alta Disponibilidade
- Escalabilidade fácil
- Flexibilidade
 - *MongoDB* armazena dados em documentos *JSON* que fornece um modelo de dados rico que mapeia perfeitamente aos tipos de linguagem de progra-

mação nativa, e o esquema dinâmico torna mais fácil evoluir seu modelo de dados do que com um sistema com esquemas forçados.

- Velocidade
 - Ao manter os dados relacionados juntos em documentos, consultas podem ser muito mais rápidas do que em um banco de dados relacional, onde os dados relacionados são separados em várias tabelas e depois precisam ser unidos mais tarde.
- Facilidade de uso
 - *MongoDB* trabalha duro para ser muito fácil de instalar, configurar, manter e usar. O *MongoDB* funciona direito fora da caixa, podendo ir direto ao desenvolvimento do aplicativo, em vez de gastar muito tempo configurando o banco de dados.

Camara & Garcia (2011) sugere que os seguintes recursos do *MongoDB* sejam considerados:

- ✓ Binários disponíveis para *Linux*, *Sun Solaris*, *Apple MacOS* e *Microsoft Windows*;
- ✓ Documentação abrangente e detalhada – diminuindo assim a curva de aprendizagem;
- ✓ Suporte a expressões regulares em consultas – Desta forma permite a realização de consultas mais complexas;
- ✓ Escala horizontal com *auto-sharding* – Permite que clusters sejam contados de forma dinâmica através de recursos oferecidos pela *API* do banco de dados *MongoDB*;
- ✓ Opções de filtragem, agregação e classificação, tais como *limit()*, *skip()*, *sort()*, *count()*, *distinct()* e *group()* – Recursos comuns no modelo relacional de banco de dados;
- ✓ Replicação *Master/Slave* – “A leitura torna-se mais rápida, porém a capacidade de escrita torna-se um gargalo nesta abordagem” (LÓSCIO, 2011, p. 4).

Memcached

O *Memcached* é fonte livre e aberta, de alto desempenho, sistema de objetos distribuídos de memória cache, de natureza genérica, mas que se destina para uso em acelerar aplicações web dinâmicas, aliviando a carga do banco de dados.(MEMCACHED, 2013)

O *Memcached* utiliza memória de armazenamento de chaves de valor para pequenos pedaços de dados arbitrários. *Memcached* é simples, mas poderoso. Seu design simples promove rápida implantação, facilidade de desenvolvimento, e resolve muitos problemas que enfrentam grandes esconderijos de dados.

CouchDB

Como citado por Eloy (2009), o termo “*Couch*” é um acrônimo de “*Cluster of Unreliable Commodity Hardware*” (Conjunto de Hardware Commodity Não-Confíáveis). Esse sistema disponibiliza uma série de características que permitem a sua utilização de forma viável em *hardware commodity*, ou seja, em máquinas utilizadas como servidores que possuem *hardware* de baixa qualidade ou antigos.

Logo, mesmo quando executado em um hardware que esteja suscetível a falhas, o banco de dados *CouchDB* tem como objetivo prover uma alta disponibilidade e confiabilidade, mesmo que eventual, assim como ser extremamente escalável (ELOY, 2009).

O *CouchDB* foi desenvolvido para a *web*, onde se torna acessível através de uma *API* denominada *REST*, que trabalha com dados utilizando *JSON* e utiliza o *HTTP* como parte integrante de sua arquitetura, (ELOY, 2009)

Com o *CouchDB*, é possível armazenar documentos com *JSON*, acessar documentos com seu navegador web via *HTTP*, consultar, combinar e transformar documentos com JavaScript. O *CouchDB* funciona também em aplicativos móveis.(COUCHDB, 2013)

O *CouchDB* vem com um conjunto de recursos, tais como *on-the-fly* (transformação de documentos e notificações em tempo real de mudança) o que torna mais fácil o desenvolvimento *Web App*, com administração web fácil de utilizar.

Principais características:

- Dimensionamento distribuído

- Altamente disponível
- Partição tolerante
- Consistente
- Armazenamento tolerante a falhas

Voldemort

Voldemort é um sistema de armazenamento distribuído de valor-chave. (VOLDEMORT, 2013)

Principais características:

- Os dados são automaticamente replicados em vários servidores.
 - Os dados são automaticamente divididos para que cada servidor contenha apenas um subconjunto do total de dados.
- Falhas do servidor são tratadas de forma transparente

Ele é usado no *LinkedIn* para determinados problemas de escalabilidade, alta de armazenamento onde o particionamento funcional simples não é suficiente. (VOLDEMORT, 2013)

Basex

Basex é uma solução escalável e de alto desempenho, oferece uma arquitetura cliente / servidor poderosa para lidar com leitura e gravação de operações simultâneas de vários usuários. Um *frontend* visual apresenta várias visualizações hierárquicas para explorar seus dados. (BASEX, 2013)

Principais características

- Armazenamento de alto desempenho de banco de dados com os índices de texto, atributo completo e caminho.
- Apoio eficiente das *XPath / XQuery W3C*. Recomendações e texto completo e extensões de Atualização.
- Uma das mais altas taxas de conformidade disponíveis para todas as especificações suportadas.

- Arquitetura cliente / servidor, suportando ACID transações seguras, gerenciamento de usuários.
- Visualizações altamente interativas.

Amazon SimpleDB

O *Amazon SimpleDB* é um banco de dados com armazenamento de dados altamente disponível, flexível e não relacional que minimiza o trabalho da administração do banco de dados. Os desenvolvedores simplesmente armazenam e consultam itens de dados por meio de solicitações de serviços da *Web* e o *Amazon SimpleDB* faz o restante. (AMAZON SIMPLEDB, 2013)

Desvinculado pelos requisitos exigentes de um banco de dados relacional, o *Amazon SimpleDB* foi otimizado para fornecer alta disponibilidade, flexibilidade e facilidade de escalabilidade com pouca ou nenhuma carga administrativa. O *Amazon SimpleDB* cria e gerencia várias réplicas distribuídas geograficamente dos seus dados de forma automática para permitir a alta disponibilidade e a durabilidade dos dados. O serviço cobra somente pelos recursos realmente consumidos no armazenamento dos dados e no atendimento das solicitações. É possível alterar o modelo de dados durante o processo e os dados serão automaticamente indexados. Com o *Amazon SimpleDB*, é possível concentrar no desenvolvimento de aplicativos sem se preocupar com o provisionamento da infraestrutura, alta disponibilidade, manutenção do software, gerenciamento de esquemas e de índices, ou adequação do desempenho.

O *Amazon SimpleDB* fornece uma interface simples de serviços *web* para criar e armazenar dados múltiplos, consultar dados facilmente e retornar os resultados. Os dados são posicionados automaticamente, tornando mais fácil localizar rapidamente as informações. Não há necessidade de pré-definir um esquema ou alterar um esquema se novos dados serão adicionados mais tarde, redimensionar é tão simples quanto criar novos domínios, em vez de construir novos servidores.

Cassandra

O banco de dados *Apache Cassandra* tem com características principais a escalabilidade e alta disponibilidade, sem comprometer o desempenho. Escalabilidade linear e comprovada tolerância a falhas em hardware ou infraestrutura de nuvem torna a plataforma perfeita para dados de missão crítica. *Cassandra* possui apoio para replicar em vários *datacenters*, é o

melhor na sua classe, proporcionando menor latência para seus usuários. (CASSANDRA, 2013)

No *Cassandra* o modelo de dados oferece a conveniência de índices de coluna, suporte forte para desnormalização - o processo de tentar otimizar o desempenho de leitura (ou consultas) de um banco de dados, adicionando dados redundantes - e visões materializadas e poderoso *built-in* cache.

Banco de dados utilizado

Em um primeiro momento o banco de dados que seria utilizado era o *CouchDB*, por todas as suas características que favoreciam o desenvolvimento da aplicação, contudo o *plugin* do *GRAILS* para esse banco de dados está há muito tempo desatualizado e sem manutenção, isso dificultou sua utilização, ao adicionar o *plugin* a aplicação parava de funcionar por completo.

Depois destes problemas, foi decidido que o banco de dados utilizado seria o *MongoDB*, um banco ideal para aplicações web, que atende os propósitos da aplicação.

O *MongoDB* é um banco de dados orientado a documentos de alta performance, *open source* de esquema livre, formado por uma mistura entre os repositórios escaláveis e com tradicional riqueza das funcionalidades dos bancos de dados relacionais. (MATTEUSSI, 2010)

Segundo os desenvolvedores do banco de dados *MongoDB* o seu principal objetivo é fechar a lacuna entre o rápido e altamente escalável modelo chave-valor e os bancos de dados relacionais que são ricos em funcionalidades tradicionais (STRAUCH, 2013).

Quando se é preciso aplicar uma solução que utilize banco de dados orientado a documentos, o *MongoDB* pode ser uma boa opção, pois além de apresentar uma documentação abrangente e detalhada, também oferece recursos avançados de consulta (como filtragem, agregação e classificação), o que torna a curva de aprendizagem menor para desenvolvedores que têm experiência na utilização do modelo relacional.

2.2 JSON (JavaScript Object Notation)

JSON (*JavaScript Object Notation*) é um formato de intercâmbio de dados leve. É baseado em um subconjunto da linguagem de programação *JavaScript*, padrão ECMA-262 3ª Edição - dezembro 1999. *JSON* é um formato de texto que é completamente independente do

idioma, mas usa convenções que são familiares aos programadores de C, C++, C#, Java, JavaScript, Perl, Python, entre outras. Estas propriedades fazem *JSON* uma linguagem de intercâmbio de dados ideal.

Os bancos de dados *NOSQL*, como o *MongoDB*, utilizam a notação *JSON*, porque a estrutura de dados fica mais simples de trabalhar e o tempo de execução de um script lendo dados em *JSON* é dezenas de vezes mais rápido do que ler um conteúdo *XML*, é também o modelo de dados adotado pelo *Elasticsearch*.

JSON é construído em duas estruturas: Uma coleção de pares nome / valor. Em várias línguas, este é percebido como um *objeto*, registro, *struct*, dicionário, tabela *hash*, lista com chave, ou *array* associativo. Uma lista ordenada de valores. Na maioria das linguagens, isto é percebido como um *array*, vetor, lista ou sequência.

Um *objeto*, representado pela Figura 1, é um conjunto desordenado de pares nome/valor. Um objeto começa com { (chave esquerda) e termina com } (chave direita). Cada nome é seguido por : (dois pontos) e os pares nome/valor são separados por , (vírgula).

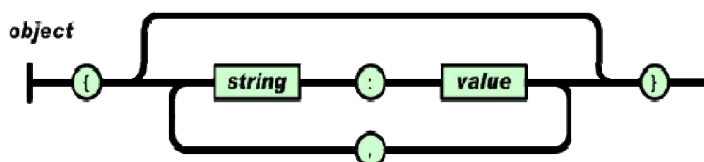


Figura 1: Objeto (JSON, 2013)

Uma *matriz*, representada pela Figura 2, é uma coleção ordenada de valores. Um *array* começa com [(colchete esquerdo) e termina com] (colchete direito). Os valores são separados por , (vírgula).

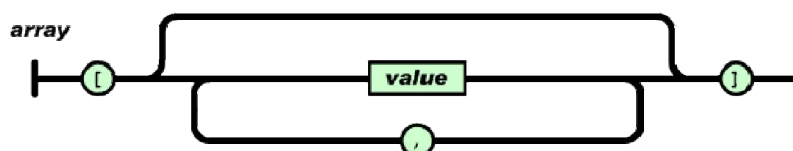


Figura 2 : Matriz. (JSON, 2013)

Um *valor*, representado pela Figura 3, pode ser uma string entre aspas, ou um número, ou verdadeiro ou falso ou nulo, ou um objeto ou um array. Estas estruturas podem ser aninhadas.

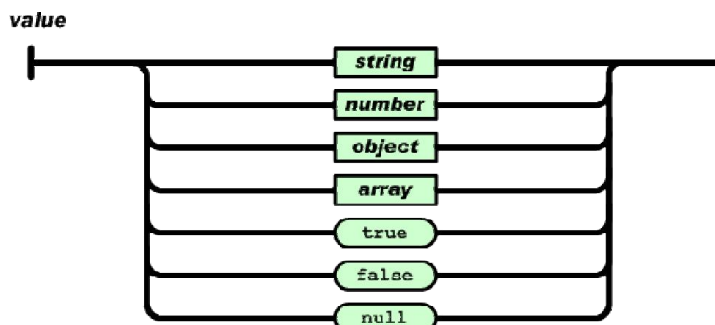


Figura 3: Valor. (JSON, 2013)

Uma *string*, representado pela Figura 4, é uma sequência de zero ou mais caracteres Unicode, envolto em aspas duplas, usando escapes. Um personagem é representado por uma única cadeia de caracteres.

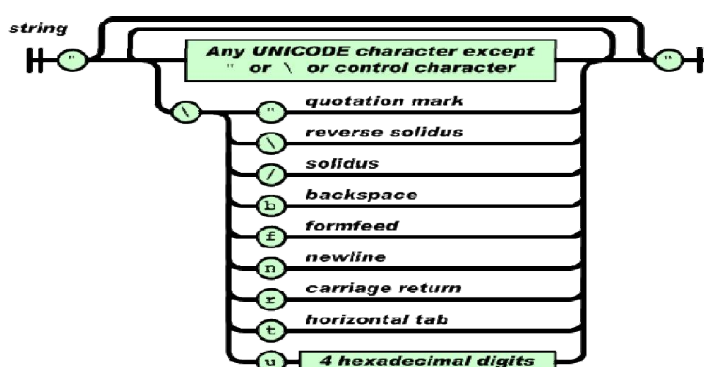


Figura 4: String. (JSON, 2013)

Um *número*, representado pela Figura 5, é muito parecido com uma série C ou Java, exceto que o octal e formatos hexadecimais não são usados.

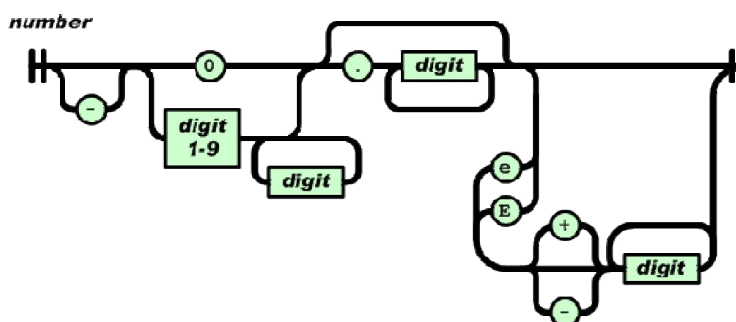


Figura 5: Número. (JSON, 2013)

2.3 Elasticsearch

As ferramentas de busca podem ser definidas como programas ou sites especializados em localizar e/ou descrever informações de arquivos, documentos, por exemplo, ou seja, é um instrumento de pesquisa através de palavras-chave ou categorias, orientadas por texto. Esses programas são também conhecidos como Mecanismos de Busca, *Search Engines*, etc. Um desses mecanismos de busca é o *Elasticsearch*.

O *Elasticsearch* provê uma busca textual para aplicações, que tem se tornado cada vez mais fácil com as máquinas de busca *open source* disponíveis na internet. O *elasticsearch*, por exemplo, é uma máquina de busca cujo modelo de dados é baseado em banco de dados *schema-free* (não é necessário definir a estrutura dos dados a priori) e orientado a documentos. O modelo adotado pelo *elasticsearch* é o *JSON* e a máquina por trás é o poderoso *Apache Lucene*, um texto de pesquisa da biblioteca motor de alta performance, com recursos completos escrito inteiramente em *Java*, é uma tecnologia adequada para quase qualquer aplicação que requer pesquisa de texto completo, construção de filtros de pesquisa, etc. (Anexo I). (ELASTICSEARCH, 2013)

Uma das principais características do *elasticsearch* que chama atenção é o fato dele ser distribuído o que permite uma fácil escalabilidade tanto vertical como horizontal e alta disponibilidade.

2.3.1 Características do Elasticsearch

Distribuído *Elasticsearch* permite começar pequeno, mas cresça com o negócio. Ele é construído para escalar horizontalmente (*scale out*), ou seja, significa adicionar mais nós ao sistema. À medida que precisar de mais capacidade, basta adicionar mais nós, e deixar o cluster se reorganizar para aproveitar o *hardware* extra.

A alta disponibilidade Cachos *Elasticsearch* são resistentes - eles vão detectar e remover nós com falha, e se reorganizar para assegurar que seus dados estão seguros e acessíveis.

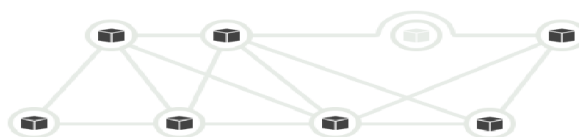


Figura 6: Cachos ElasticSearch (ELASTICSEARCH, 2013)

Pesquisa de texto completo: o *Elasticsearch* usa *Lucene* por trás para fornecer os mais poderosos recursos de pesquisa de texto completo disponíveis em qualquer produto de código aberto. A pesquisa vem com suporte a multilinguagem, uma poderosa linguagem de consulta, apoio à geolocalização.

Orientado a Documentos: Armazena entidades do mundo real complexas em *Elasticsearch* como documentos *JSON* estruturados. Todos os campos são indexados por padrão, e todos os índices podem ser usados em uma única consulta, para retornar resultados rapidamente.

Esquema grátis: O *Elasticsearch* permite lançar um documento *JSON*, tenta detectar a estrutura de dados, o índice dos dados e torna-o pesquisável. Depois, aplicar o domínio do conhecimento específico de seus dados para personalizar o modo como serão indexados.

2.4 Sphinx (Esfinge)

Sphinx é um completo servidor *open source* de pesquisa, concebido a partir do zero com o desempenho, relevância, e simplicidade de integração em mente. Ele é escrito em C++ e funciona em Linux (RedHat, Ubuntu, etc), Windows, MacOS, Solaris, FreeBSD, e alguns outros sistemas.(SPHINIX, 2013)

2.4.1 Características da Sphinx:

- **Sintaxe pesquisa de texto completa avançada.** O motor de busca suporta consultas arbitrariamente complexas combinando operadores booleanos, frase de proximidade, de ordem estrita, de campo e de limites de posição, forma exata palavra-chave correspondente, pesquisas substring, etc.
- **Melhor classificação de relevância.** Ao contrário de muitos outros motores, a *Sphinx*, adicionalmente, analisa palavra-chave de proximidade, e ocupa a frase mais próxima corresponde superior, com combinações perfeitas classificados no topo. Além disso, a classificação é flexível: é possível escolher a partir de uma série de funções de relevância *built-in*, ajustar seus pesos usando expressões.
- **Buscas Distribuídas.** Pesquisas podem ser distribuídas em várias máquinas, permitindo horizontal *scale-out* e HA (*High Availability*).

- **Indexação e armazenamento NOSQL.** Dados também podem ser transmitidos para indexador lote em um formato XML simples chamado *XML pipe*, ou inserido diretamente em um índice RT incremental.

2.5 Groovy

Groovy é uma linguagem dinâmica para a Máquina Virtual Java, baseia-se nos pontos fortes do *Java*, mas tem características de alimentação adicionais inspirado por linguagens como *Python*, *Ruby* e *Smalltalk*. Fornece a capacidade de estaticamente verificar tipos e estaticamente compilar o código com robustez e desempenho. Suporta linguagens específicas de domínio e outras sintaxes compactas para que o código torne-se fácil de ler e manter. Faz escrever *shell* scripts de construção com as suas primitivas de processamento poderosos, habilidades OO. Aumenta a produtividade do desenvolvedor, no desenvolvimento *web*, *GUI*, banco de dados ou aplicativos de *console*. Simplifica os testes, apoiando os testes de unidade, integra com todas as classes e bibliotecas *Java* existentes, compila diretamente para *bytecode* Java. (GROOVY, 2013)

Exemplos de códigos Groovy:

Um *script* simples Olá Mundo:

```
nome def = 'Mundo'; println "Olá $ name"
```

Uma versão mais sofisticada utilizando Orientação a Objetos:

```
Cumprimente classe {
  nome def
  Greet (que) {name = que [0]. ToUpperCase () +
               que [1 .. -1]}
  def saudação () {println "Olá $ nome!" }
}

g = new Greet ('mundo') // criar o objeto
g.salute () // output "Olá mundo!"
```

Aproveitando bibliotecas Java existentes:

```
importar org.apache.commons.lang.WordUtils estáticas. *

classe Greeter estende Greet {
  Greeter (quem) {name = capitalizar (OMS)}
}
```

```
new Greeter ("mundo"). saudação ()
```

2.6 GRAILS

GRAILS é um *framework* para construção de aplicações para web através da linguagem de programação *Groovy* (uma linguagem dinâmica para a plataforma Java). Foi inicialmente chamado de "*Groovy on Rails*" até ser renomeado para *Grails*, após um pedido do fundador do projeto *Ruby on Rails*, David Heinemeier Hansson. Os trabalhos iniciaram em julho de 2005 e a versão 0.1 foi liberada em março de 2006. O *framework* isola o desenvolvedor dos detalhes complexos da persistência de dados e incorpora o padrão de desenvolvimento *MVC* de maneira natural. Ele também fornece *templates web* para fácil implementação da interface com o usuário e suporte para programação em *Ajax*. (GRAILS, 2013)

A criação de aplicações *web* em *Java* tradicionalmente envolve a configuração de ambientes e *frameworks* do início ao fim do desenvolvimento. Esta configuração normalmente reside em arquivos *XML* que isolam estas questões do código da aplicação. Apesar desta abordagem trazer vantagens, alguns acreditam que a tarefa de criar e manter estes arquivos custa muito do tempo do desenvolvedor de aplicações.

Ao invés de exigir a utilização de uma série de arquivos *XML* o *Grails* utiliza a programação por convenção para definir o papel das várias entidades de uma aplicação. Por exemplo, uma classe cujo nome termina com *CONTROLLER* (como *ItemController*) é considerado um *controller web* (o 'C' do padrão MVC).

Lidar com o *Grails* serve como diferencial para elaborar desde projetos complexos aos mais simples. E significa também estar integrado a uma tendência de mercado cuja exigência é de alta capacidade de dinamismo. O uso do *Grails* como solução faz com que o programador possa otimizar seu tempo e distribuí-lo melhor em tarefas qualitativas, sendo simples, direto e intuitivo desenvolver com *Grails*.

2.7 GGTS (Groovy/Grails Tool Suite)

O *Groovy/Grails Tool Suite* TM (GGTS) fornece um ambiente de desenvolvimento *Eclipse-powered* para a construção de aplicações de *Groovy* e *Grails*. GGTS fornece suporte para as últimas versões de *Groovy* e *Grails*, e vem em cima dos últimos lançamentos do *Eclipse*. (GRAILS, 2013)

É incluso com GGTS a edição de desenvolvedor *vFabric tc Server*, o substituto para o *Apache Tomcat*, que oferece uma visualização gráfica em tempo real de métricas de desempenho de aplicativos que permite aos desenvolvedores identificar e diagnosticar problemas de seus *desktops*. (GRAILS, 2013)

GGTS suporta aplicação de direcionamento para servidores locais, virtuais e em nuvem. Ele está disponível gratuitamente para o desenvolvimento e uso de operações comerciais internas sem limite de tempo.

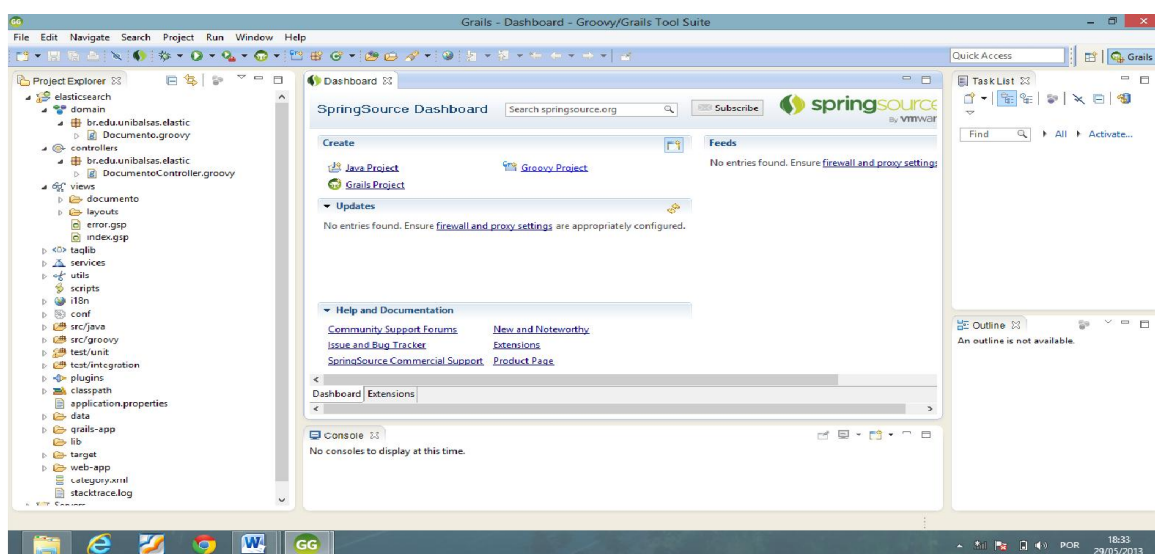


Figura 7: Ambiente GGTS

2.8 Exemplo aplicação GRAILS construída com GGTS

2.8.1 Criação do Domínio

Uma classe de domínio é um artefato persistente, e por padrão, todas as suas propriedades são persistidas no banco de dados.

```
class Livro {
    String titulo
    String autor
}
```

Neste ponto é possível criar alguns dados de teste. Uma forma fácil de fazer isso é criando e salvando os objetos de domínio na *closure "init"* da classe *bootstrap* da aplicação *Grails*, como a seguir:


```
class BootStrap {
  def init = { servletContext ->

    // Cria alguns dados de teste

    new Livro(autor:"Stephen King",titulo:"The Shining").save()

    new Livro(autor:"James Patterson",titulo:"Along Came a Spider")

    .save()

  }

  def destroy = {

  }

}
```

2.8.2 Criação do Controlador

Os controladores são centrais para as aplicações em *Grails*, eles tratam as requisições web e URLs do mapa de requisições para uma classe de controlador em uma *closure* junto a classe.

Abra este controlador e altere-o como mostrado a seguir para que seja utilizado o *Scaffolding* dinâmico, o qual gerará dinamicamente a aplicação em tempo de execução:

```
class LivroController {
  def scaffold = Livro
}
```

2.8.3 Adicionando o Plugin Elasticsearch à aplicação

Para adicionar um *plugin* a uma aplicação no GGTS, basta adicionar ao *BuildConfig.groovy* do projeto o nome do *plugin* desejado entre aspas duplas precedido da palavra *compile* e estar conectado à internet para baixá-lo e verificar todas as dependências.

Exemplo:

```
compile "elasticsearch:0.17.8.1"
```

2.9 Apache POI

O *Apache POI* é uma *API Java* utilizada para ler e escrever documentos *Word*, *Excel*, *PowerPoint*, entre outros, tendo como utilização principal a extração de texto de aplicações, como construtores de índices e sistemas de gerenciamento de conteúdo. (APACHE POI, 2013)

2.9.1 Apache POI - HWPF - Java API para lidar com arquivos do Microsoft Word

HWPF é o nome da biblioteca do POI para o formato de arquivo *Microsoft Word*. *HWPF API* fornece "ponteiros" para partes do documento, como seções, parágrafos e execuções de caracteres. O principal ponto de entrada para *HWPF* é *HWPFDocument*, que define o arquivo como .doc.

2.10 Arquitetura do Sistema de Busca e Indexação de documentos

A Figura 8 representa a arquitetura do Sistema de Busca e Indexação de documentos.

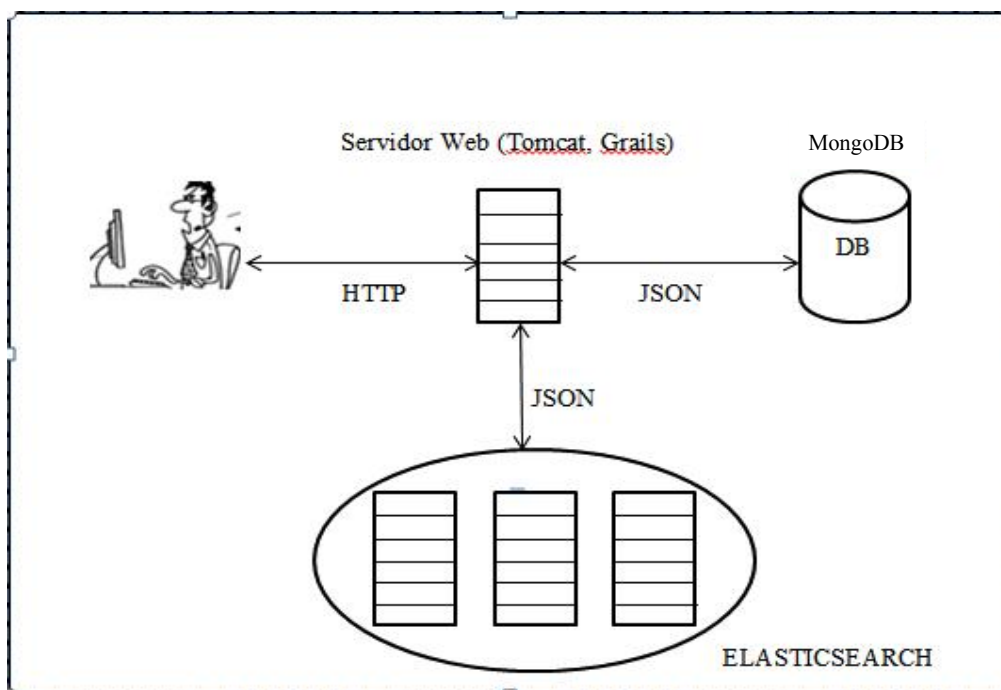


Figura 8: Arquitetura do Sistema de Busca e Indexação de Documentos

O *ElasticSearch* deverá permitir que seja feita a indexação incremental de maneira eficiente e deverá produzir um índice que seja relativamente pequeno em comparação ao tamanho total dos documentos indexados. Além disso, o índice deve armazenar informações suficientes para permitir busca por campos específicos dos documentos.

O *Tomcat* é um servidor web Java, mais especificamente, um container de *servlets*¹. Desenvolvido pela *Apache Software Foundation*, é distribuído como software livre dentro do conceituado projeto *Apache Jakarta* como a implementação de referência para as tecnologias *Java Servlet* e *JavaServer Pages* (JSP). (TOMCAT, 2013)

O Banco de Dados NOSQL, *MongoDB*, armazena os documentos numa base de índices diretos, permitindo assim a fácil recuperação do seu conteúdo. A comunicação entre o Servidor *Web* e Banco de Dados, assim como entre Servidor *Web* e *Elasticsearch*, será feita por meio do *JSON*, que permite um intercâmbio de dados leve. O usuário se comunicará com o Servidor *Web* por meio do protocolo *HTTP* (Protocolo de Transferência de Hipertexto) .

2.11 Requisitos do Sistema de Busca e Indexação de documentos

A Tabela 2 apresenta os requisitos que foram atendidos no Sistema de Busca e Indexação de documentos.

Nº Requisito	Descrição do Requisito
01	Permitir indexar documentos a partir de diversas fontes (Sistemas de arquivos, Banco de Dados)
02	Indexar o formato de documento .doc
03	Reconhecer meta-dados nos documentos, como informações sobre o autor do documento e palavras-chave.
04	Possibilitar busca por palavras em campos específicos do documento (por exemplo título, palavras-chave).
05	Apresentar para o usuário o documento pesquisado.

Tabela 2: Requisitos do Sistema de Busca e Indexação de documentos

1: Classes Java utilizadas para estender as funcionalidades de um servidor

3. DocIndex - Sistema de indexação e busca de documentos

Neste capítulo serão abordadas as práticas utilizadas para desenvolvimento do sistema DocIndex.

3.1 Configuração do banco de dados MongoDB

A configuração do banco de dados é feita no arquivo *DataSource.groovy* da seguinte maneira:

```
mongo {  
    host = "localhost"  
    port = 27017  
    username = "mongo"  
    password = ""  
    databaseName = "documento"  
}
```

3.2 Instalação do Elasticsearch no Windows

Depois de feito o download do *Elasticsearch* os arquivos foram extraídos para o C: após extrair, pelo *prompt* de comando é chamado o comando **elasticsearch.bat**, sendo assim executado o arquivo em lotes e iniciado o *Elasticsearch*, publicando o serviço no endereço 192.168.1.15:9300.

O *Elasticsearch* possui um mecanismo de auto descoberta, o próprio descobre outros serviços *Elasticsearch* ativos na rede, tornando-se assim um serviço distribuído, pois se um nó falhar o outro assumirá o serviço e a aplicação não falhará, garantindo assim sempre alta disponibilidade.

3.3 Plugins instalados

A adição de *plugins* à aplicação é feita no arquivo *BuildConfig.groovy*, neste caso foram adicionados os *plugins* para o banco de dados, *Elasticsearch*, e para fazer a leitura dos arquivos .doc, como segue abaixo respectivamente.

```
compile ":mongodb:1.3.0"  
compile ":elasticsearch:0.17.8.1"  
compile 'org.apache.poi:poi:3.9'
```

3.4 Domínio

A classe de domínio representa o modelo do núcleo da aplicação e normalmente é mapeado para tabelas de banco de dados. Os domínios criados para a aplicação foram *br.edu.unibalsas.Documento*, o qual gerou a classe de domínio *Documento.groovy*, e *br.edu.unibalsas.Paragrafo*, gerando assim a classe de domínio *Paragrafo.groovy*.

3.4.1 Documento.groovy

O domínio Documento define os atributos da classe, o mapeamento dos documentos para o banco de dados *MongoDB*, aciona o mecanismo de busca e faz restrições ao mesmo, realiza o mapeamento dos parágrafos para o documento, embarca os parágrafos em outros formando o documento.

```
package br.edu.unibalsas

import java.util.ArrayList;

class Documento {

    static mapWith = "mongo"
    static searchable = {
        except = 'caminhoServidor'
        paragrafos component:true

    static constraints = {
        paragrafos cascade:'all'
        nomeArquivoOriginal(blank:false,nullable:false)
        caminhoServidor(blank:false,nullable:false)
    }

    static hasMany = [
        paragrafos:Paragrafo
    ]

    static mappedBy = [
        paragrafos:'documento'
    ]

    static embedded = [
        'paragrafos'
    ]

    String nomeArquivoOriginal
    String autor
    String caminhoServidor
    Date dataCriacao
    Date dataAtualizacao
    String tipoArquivo
    List paragrafos
}
```

3.4.2 Paragrafo.groovy

O domínio Parágrafo define o atributo da classe, o mapeamento dos parágrafos para o banco de dados *MongoDB* e permite que eles sejam pesquisáveis.

```
package br.edu.unibalsas

import java.util.Date;

class Paragrafo{

    static mapWith = "mongo" // mapeia o documento para o banco de dados MongoDB

    static searchable = { faz com que o documento seja buscável
        texto boost:2.0
        root false
    }
    static belongsTo = [
        documento:Documento
    ]

    static constraints = {
        texto nullable:true,cascade:'all'
    }

    // Atributo da classe
    String texto = ''
}
```

3.5 Controladores

Um controlador é responsável pelo tratamento de pedidos de entrada da *Web* e executar ações, como redirecionamentos, vistas de renderização e assim por diante. No ambiente GGTS, o comando *generate-controller*, gera automaticamente o controlador da classe de domínio trazendo já implementadas *actions*, como *create*, *delete*, *edit*, *save list* e *show*, basta que sejam modificadas de acordo com a necessidade da aplicação, no caso do *DocIndex*, foram alteradas as *actions*, *save* e *delete*, e criadas, *search* e *download*.

3.5.1 Action Save

A *action save*, salva o arquivo em uma pasta o servidor, cria um índice com o *elasticsearch* e mapeia o índice para o banco de dados *MongoDB*.

Alguns comandos utilizados nesta *action*:

```

def documentoInstance = new Documento(params)// traz os atributos do Documento
def f = request.getFile('arquivo')// renderiza para a action uploadForm para fazer
o upload do arquivo.
    if (f.empty){
        flash.message = 'file canot be empty'
        render (view: 'uploadForm')return}
// Transfere o arquivo para a pasta do servidor
def file = new File('C:/documentos/' + documentoInstance.nomeArquivoOriginal +
'.doc')
f.transferTo(file)

// este trecho faz a leitura dos parágrafos do documento e define listas de para-
gros associadas ao documento (Utilização do plugin Apache POI)
FileInputStream fis = new FileInputStream(file);

    WordExtractor extractor = null;

    HWPFDocument document = new HWPFDocument(fis);
    extractor = new WordExtractor(document);
    String [] fileData = extractor.getParagraphText();
    for(int i=0;i<fileData.length;i++){

        if(fileData[i] != null){
            def paragrafo = new Paragrafo(texto:fileData[i])
            paragrafos << new Paragrafo(texto:fileData[i])
        }

    }

//cria um índice para a instancia do documento
elasticSearchService.index(documentoInstance);

```

3.5.2 Action Delete

A *action Delete* é gerada automaticamente, nesta aplicação foi acrescentado apenas um comando para destruir o índice gerado pelo *Elasticsearch* para o documento.

```
elasticSearchService.unindex(documentoInstance)
```

3.5.3 Action Search

A *action search* é responsável por realizar a busca dos documentos a partir de *queries*, estas podem ser palavras-chave, nome do arquivo, nome do autor.

```

def search (){
    println params
    def documentoInstanceList = [] //lista de documentos vazia
    def q = params.q // parametros do documento
    println q
    def total = 0 // total de documentos encontrados
    if ( q != null ){
        def res = Documento.search("${q}")// passa a query para a busca
        println "Found ${res.total} result (s)"
    }
}

```

```

        res.searchResults.each {
            if (it instanceof Documento){
                documentoInstanceList << it //passa a lista de do-
cumentos
            }
        }

        total = res.total // total de documentos encontrados

    }else {
        q = "" // se nada for encontrado retorna vazio
    }

    [documentoInstanceTotal: total, documentoInstanceList: documentoIns-
tanceList, q: q ]
}

```

3.5.4 Action Download

A *action Download* permite fazer o download dos arquivos armazenados pela aplicação na pasta *documentos* do servidor.

```

def download(long id) {
    Documento documentoInstance = Documento.get(id)

    if ( documentoInstance == null) {
        flash.message = "Document not found."
        redirect (action:'list')
    } else {
        response.setContentType("APPLICATION/OCTET-STREAM")
        response.setHeader("Content-Disposition", "Attach-
ment;Filename=\"${documentoInstance.nomeArquivoOriginal}\"")
        def file = new File('C:/documentos/' + documentoInstan-
ce.nomeArquivoOriginal + '.doc')
        def fileInputStream = new FileInputStream(file)
        def outputStream = response.getOutputStream()

        byte[] buffer = new byte[4096];
        int len;
        while ((len = fileInputStream.read(buffer)) > 0) {
            outputStream.write(buffer, 0, len);
        }

        outputStream.flush()
        outputStream.close()
        fileInputStream.close()
    }
}

```

3.5.5 Action UploadForm

A *action uploadForm* é responsável por fazer o upload dos arquivos para a aplicação.


```

def file = new File('C:/documentos/' + documentoInstance.nomeArquivoOriginal +
'.doc')
    f.transferTo(file)
    FileInputStream fis = new FileInputStream(file);
    WordExtractor extractor = null;
    HWPFDocument document = new HWPFDocument(fis);
    extractor = new WordExtractor(document);
    String [] fileData = extractor.getParagraphText();
    for(int i=0;i<fileData.length;i++){
        if(fileData[i] != null){
            def paragrafo = new Paragrafo(texto:fileData[i])
            paragrafos << new Paragrafo(texto:fileData[i])
        }
    }
}


```

3.6 Views

As views representam o ‘V’ do padrão MVC, e ficam encarregadas por apresentar ao usuário a aplicação. A maioria das *views* são geradas automaticamente pelo comando *generate-views*, entre elas estão, *create*, *list*, *show*, *_form* para cada controlador, nesta aplicação algumas dessas forma modificadas e outras criadas, *search* e *uploadForm*. (Apêndice I)

3.7 Telas do Sistema

A Figura 9 apresenta a primeira tela do Sistema DocIndex, nesta tela o usuário visualiza todos os documentos que estão armazenados pela aplicação.



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/Documento/documento/list'. The page title is 'DocIndex - Sistema de Busca e Indexação de Documentos'. Below the title is a navigation bar with links: 'Principal', 'Novo Documento', and 'Pesquisar Documento'. The main content area is titled 'Documento Listagem' and contains a table with the following data:

Autor	Nome do Arquivo	Data Atualizacao	Data Criacao	Tipo Arquivo	Download
Igor	Monografia Parcial	26/11/2013 23:44:52 GMT-03:00	26/11/2013 23:44:52 GMT-03:00	WORD	Monografia Parcial
Igor	ILTIL	26/11/2013 23:53:46 GMT-03:00	26/11/2013 23:53:46 GMT-03:00	WORD	ILTIL
Igor	Artigo	26/11/2013 23:54:38 GMT-03:00	26/11/2013 23:54:38 GMT-03:00	WORD	Artigo
Antonio	Sistemas	26/11/2013 23:55:23 GMT-03:00	26/11/2013 23:55:23 GMT-03:00	WORD	Sistemas
Igor	Artigo	28/11/2013 16:00:29 GMT-03:00	28/11/2013 16:00:29 GMT-03:00	WORD	Artigo

At the bottom of the page, there is a status bar that says 'Aguardando localhost...'.

Figura 9: Listagem de Documentos

A Figura 10 representa ao usuário a tela de Criação de um novo documento. Onde o usuário informa o nome do autor do documento, nome do arquivo e faz o upload do arquivo para salvá-lo no servidor da aplicação.

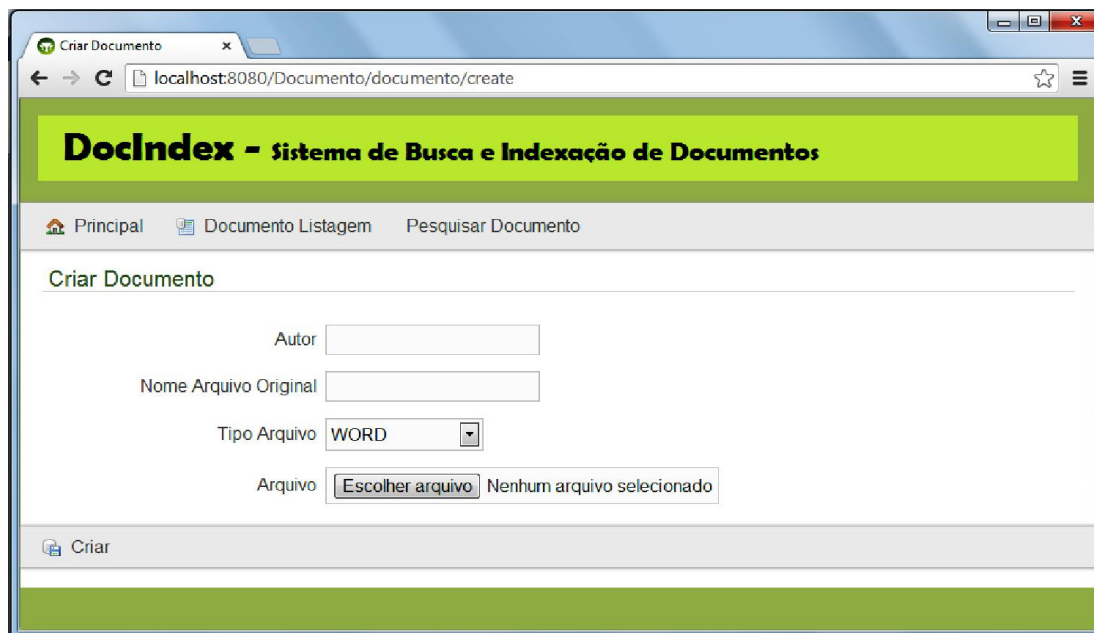


Figura 10: Criação de um novo Documento

A Figura 11 representa a tela de exibição dos detalhes do documento.




Figura 11: Tela Show Documento

A Figura 12 representa a tela de busca da aplicação, nesta tela é possível realizar busca por documentos e visualizar as informações do documento buscado e fazer seu download. É possível realizar busca pelo nome do autor, do arquivo ou por palavras-chave.

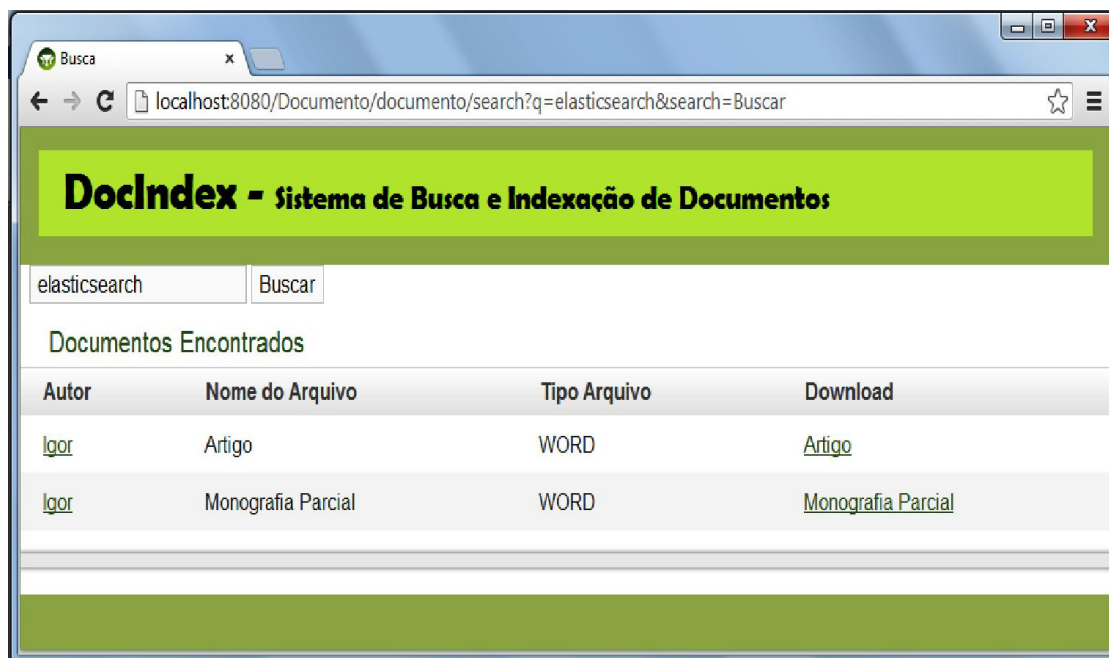


Figura 12: Tela de Busca

3.8 Dificuldades encontradas

Uma das principais dificuldades encontradas foi em relação aos *Plugins*, que na maioria das vezes estavam desatualizados. Essa foi a principal razão para que houvesse a mudança do banco de dados *CouchDB* para o *MongoDB*. O *plugin* do *Apache POI* apresentou alguns problemas, como documentação desatualizada, isso acabou resultando para que a aplicação nesta primeira versão somente indexe e busque arquivos *Word* (.doc).

A instalação do *Elasticsearch* também apresentou alguns problemas, pois a versão que o *plugin* utilizava do *Elasticsearch* estava desatualizada em relação a última lançada, tendo algumas limitações, como a de não poder definir o nó *master* e o *slave* manualmente.

Outra dificuldade encontrada foi em relação à falta de livros dos temas relacionados ao desenvolvimento deste projeto.

4. CONCLUSÃO E ATIVIDADES FUTURAS

Para atingir os objetivos deste projeto foi realizado toda uma pesquisa bibliográfica que forneceu o embasamento teórico necessário para o desenvolvimento do sistema de busca e indexação de documentos empresariais, desde aspectos relacionados a banco de dados, como conceitos, características e tipos, à indexação de documentos com o *Elasticsearch* e definição do ambiente de desenvolvimento utilizado.

Após definir as ferramentas e banco de dados que foram utilizados, foi realizado o desenvolvimento da ferramenta de indexação e busca de documentos atendendo todos os requisitos propostos, uma vez que foi desenvolvida, passou por testes e análise dos resultados da utilização do Sistema DocIndex.

Como resultado deste projeto, tem-se o Sistema DocIndex, distribuído e escalável, este armazena e recupera documentos de forma rápida garantindo a integridade das informações.

Nesta primeira versão o Sistema DocIndex conta somente com o tipo de arquivo Word (.doc), mas futuramente poderão implementados os tipos de arquivos Excell (.xls) e PDF (.pdf).

5 REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, Ricardo Cardoso de; BRITO Parcilene Fernandes de. **Utilização da classe de banco de dados NOSQL como solução para manipulação de diversas estruturas de dados**. Disponível em: <http://ulbra-to.br/encoinfo/artigos/2012/utilizacao_da_classe_de_banco_de_dados_nosql_como_solucao_para_manipulacao_de_diversas_estruturas_de_dados.pdf>. Acesso em: 28 set. 2013.

Apache Tomcat. Disponível em: <<http://tomcat.apache.org/>>. Acesso em: 18 jun. 2013.

Basex Home. Disponível em: <<http://basex.org/>>. Acesso em: 12 abr. 2013.

BRITO, Ricardo W. **Bancos de Dados NoSQL x SGBDs Relacionais: Análise Comparativa**. In: InfoBrasil, 3, 2010, Fortaleza, Ceará.

CAMARA, Evandro Augusto Muchinski; GARCIA, Renato Joukoski. **Avaliação de Sistemas de Reconhecimento de Fala Para Indexação Automática De Vídeos**. 2011. 39 p. Monografia (Bacharel em Ciências da Computação) – Universidade Federal do Paraná, Curitiba, Paraná.

Cassandra Home. Disponível em: <<http://cassandra.apache.org/>>. Acesso em: 12 abr. 2013.

CouchDB About. Disponível em: <<http://couchdb.apache.org/>>. Acesso em: 10 abr. 2013.

DIANA, Mauricio De; GEROSA, Marco Aurélio. **NOSQL na Web 2.0: Um Estudo Comparativo de Bancos Não-Relacionais para Armazenamento de Dados na Web 2.0**. In: Workshop de Teses e Dissertações em Banco de Dados, 9, 2010, Belo Horizonte, Minas Gerais.

DOLIRIO, Victor. **Introdução Ao Elasticsearch: Busca textual em sua aplicação**. Disponível em: <<http://blog.victordolirio.com/2012/10/um-google-na-sua-aplicacao-com-o.html>>. Acesso em: 26 set. 2013.

Elasticsearch. Disponível em: <www.elasticsearch.org>. Acesso em: 27 abr. 2013.

ELOY, Leonardo Abrantes de Oliveira. **O Estado da Arte em Bancos de Dados Orientados a Documentos**. 2009. 78f. Trabalho de conclusão de curso (Graduação). Universidade de Fortaleza. Fortaleza, Ceará.

GARDIN, J.C. **Les analyses des discours**. Neuchatel: Delachaux et Nestlé, 1974.

GGTS. Disponível em: <<http://www.grails.org/products/ggts>>. Acesso em: 04 maio 2013.

Grails Home. Disponível em: <<http://www.grails.org/>>. Acesso em: 28 abr. 2013.

Home Groovy. Disponível em: <<http://groovy.codehaus.org/>>. Acesso em: 28 abr. 2013.

Introducing JSON. Disponível em: <<http://www.json.org/>>. Acesso em: 27 abr. 2013.

LÓSCIO, Bernadette Farias; OLIVEIRA, Hélio Rodrigues de; PONTES, Jonas César de Sousa. **NoSQL no desenvolvimento de aplicações Web colaborativas**. In: SIMPÓSIO BRASILEIRO DE SISTEMAS COLABORATIVOS, 8, 2011, Paraty, Rio de Janeiro.

LONGHI, Fulvio. **Bancos de Dados NOSQL**: Escalabilidade de dados sem limites. Disponível em: <<http://www.slideshare.net/jornaljava/bancos-de-dados-nosql-jornaljava>>. Acesso em: 02 de Abr. 2013.

MATTEUSSI, Kassiano José. **Protótipo de Interface Web com PHP para Gerenciamento de Banco de Dados CouchDB**. 2010. 81 p. Monografia (Bacharel em Ciências da Computação) – Universidade Comunitária Da Região De Chapecó, Chapecó, Santa Catarina.

MemcachedB Guide. Disponível em: <<http://www.memcachedb.org>>. Acesso em: 10 abr. 2013.

MongoDB. Disponível em: <<http://www.mongodb.org/>>. Acesso em: 10 abr. 2013.

Neo4j. Disponível em: <<http://www.neo4j.org/>>. Acesso em: 10 abr. 2013.

PINTO, Virgínia Bentes. Indexação documentária: uma forma de representação do conhecimento registrado. **Perspect. Cienc. Inf.**, Belo Horizonte, v. 6, n. 2, P.223-234, 25 jan. 2001.

RSLIBRIS CONSULTORIA. **Você Sabia?** Disponível em: <http://www.rslibris.com.br/html/vcsabia.html>. Acesso em: 18 de Mar. 2013.

SPHINX About. Disponível em: <<http://sphinxsearch.com/about/sphinx/>>. Acesso em: 28 abr. 2013.

STRAUCH, Christof. **NoSQL Databases**. Disponível em: <http://www.christofstrauch.de/nosql dbs.pdf>. Acesso em: 26 de Set. de 2013.

Amazon SimpleDB. Disponível em: <<http://aws.amazon.com/pt/simpledb/>>. Acesso em: 12 abr. 2013.

Voldemort QuickStart. Disponível em: <<http://www.projectvoldemort.com/voldemort/>>. Acesso em: 12 abr. 2013.

ANEXO I

Elasticsearch Query

Filter:

Filter é uma consulta que aplica um filtro à consulta. Exemplo:

```
{
  "Filtered": {
    "Query": {
      "Termo": {"tag": "wow"}
    }
    "Filter": {
      "Range": {
        "Idade": {"from": 10, "to": 20}
      }
    }
  }
}
```

O objeto *Filter* pode conter apenas elementos filtrantes, e não consultas. Os filtros podem ser muito mais rápidos em comparação às consultas, uma vez que não realizam qualquer pontuação, especialmente quando eles são armazenados em cache.

APÊNDICE I

Create

Nesta view são criados os documentos, passando as informações sobre o documento, como autor, nome do arquivo e o próprio arquivo para upload, depois de fornecidas as informações a action *save* é chamada para salvar o documento no servidor.

```
<%@ page import="br.edu.unibalsas.Documento" %>
<!DOCTYPE html>
<html>
    <head>
        <meta name="layout" content="main">
        <g:set var="entityName" value="${message(code: 'documento.label', default: 'Documento')}" />
        <title><g:message code="default.create.label" args="[entityName]" /></title>
    </head>
    <body>
        <a href="#create-documento" class="skip" tabindex="-1"><g:message code="default.link.skip.label" default="Skip to content&hellip;" /></a>
        <div class="nav" role="navigation">
            <ul>
                <li><a class="home" href="${createLink(uri: '/')}"><g:message code="default.home.label" /></a></li>
                <li><g:link class="list" action="list"><g:message code="default.list.label" args="[entityName]" /></g:link></li>
                <li><g:link class="search" action="search"><g:message code="Pesquisar Documento" args="[entityName]" /></g:link></li>
            </ul>
        </div>
        <div id="create-documento" class="content scaffold-create" role="main">
            <h1><g:message code="default.create.label" args="[entityName]" /></h1>
            <g:if test="${flash.message}">
                <div class="message" role="status">${flash.message}</div>
            </g:if>
            <g:hasErrors bean="${documentoInstance}">
                <ul class="errors" role="alert">
                    <g:eachError bean="${documentoInstance}" var="error">
                        <li><g:if test="${error instanceof org.springframework.validation.FieldError}">data-field-id="${error.field}"</g:if><g:message error="${error}" /></li>
                    </g:eachError>
                </ul>
            </g:hasErrors>
            <g:uploadForm action="save" >
                <fieldset class="form">
                    <g:render template="form" />
                </fieldset>
                <fieldset class="buttons">
                    <g:submitButton name="create" class="save" value="${message(code: 'default.button.create.label', default: 'Create')}" />
                </fieldset>
            </g:uploadForm>
        </div>
    </body>
</html>
```



```

        </fieldset>
    </g:uploadForm>
</div>
</body>
</html>

```

List

Esta view é responsável por disponibilizar ao usuário a visualização de todos os documentos armazenados no servidor da aplicação, a partir desta view é possível pesquisar, criar, visualizar informações e baixar documentos.

```

<%@ page import="br.edu.unibalsas.Documento" %>
<!DOCTYPE html>
<html>
    <head>
        <meta name="Layout" content="main">
        <g:set var="entityName" value="${message(code: 'documento.label', default: 'Documento')}" />
        <title><g:message code="default.List.Label" args="[entityName]" /></title>
    </head>
    <body>
        <a href="#list-documento" class="skip" tabindex="-1"><g:message code="default.link.skip.Label" default="Skip to content&hellip;" /></a>
        <div class="nav" role="navigation">
            <ul>
                <li><a class="home" href="${createLink(uri: '/' )}"><g:message code="default.home.Label" /></a></li>
                <li><g:link class="create" action="create"><g:message code="default.new.Label" args="[entityName]" /></g:link></li>
                <li><g:link class="search" action="search"><g:message code="Pesquisar Documento" args="[entityName]" /></g:link></li>
            </ul>
        </div>
        <div id="list-documento" class="content scaffold-list" role="main">
            <h1><g:message code="default.List.Label" args="[entityName]" /></h1>
            <g:if test="${flash.message}">
                <div class="message" role="status">${flash.message}</div>
            </g:if>
            <table>
                <thead>
                    <tr>
                        <g:sortableColumn property="autor" title="${message(code: 'documento.autor.label', default: 'Autor')}" />
                        <g:sortableColumn property="nomeArquivoOriginal" title="${message(code: 'documento.nomeArquivoOriginal.label', default: 'Nome do Arquivo')}" />

```

```

                                <g:sortableColumn proper-
ty="dataAtualizacao" title="${message(code: 'documento.dataAtualizacao.label', de-
fault: 'Data Atualizacao')}"/>

                                <g:sortableColumn property="dataCriacao"
title="${message(code: 'documento.dataCriacao.label', default: 'Data Criacao')}"/>
                                />

                                <g:sortableColumn property="tipoArquivo"
title="${message(code: 'documento.tipoArquivo.label', default: 'Tipo Arquivo')}"/>
                                />

                                <g:sortableColumn property="tipoArquivo"
title="${message(code: 'documento.tipoArquivo.label', default: 'Download')}"/>
                                />

                                </tr>
                                </thead>
                                <tbody>
                                <g:each in="${documentoInstanceList}" status="i"
var="documentoInstance">
                                <tr class="${(i % 2) == 0 ? 'even' : 'odd'}">

                                <td><g:link action="show"
id="${documentoInstance.id}">${fieldValue(bean: documentoInstance, field: "au-
tor")}</g:link></td>

                                <td>${fieldValue(bean: documentoInstance,
field: "nomeArquivoOriginal")}</td>

                                <td><g:formatDate
date="${documentoInstance.dataAtualizacao}" /></td>

                                <td><g:formatDate
date="${documentoInstance.dataCriacao}" /></td>

                                <td>${fieldValue(bean: documentoInstance,
field: "tipoArquivo")}</td>

                                <td><g:link action="download"
id="${documentoInstance.id}">${documentoInstance.nomeArquivoOriginal}</g:link></td>
                                </tr>
                                </g:each>
                                </tbody>
                                </table>
                                <div class="pagination">
                                <g:paginate total="${documentoInstanceTotal}" />
                                </div>
                                </div>
                                </body>
                                </html>

```

Show

A view Show é responsável por apresentar as informações sobre o documento, a partir desta, pode se criar, pesquisar e fazer download do documento.

```
<%@ page import="br.edu.unibalsas.Documento" %>
<!DOCTYPE html>
<html>
    <head>
        <meta name="layout" content="main">
        <g:set var="entityName" value="${message(code: 'documento.label', de-
fault: 'Documento')}" />
        <title><g:message code="default.show.Label" args="[entityName]"
/></title>
    </head>
    <body>
        <a href="#show-documento" class="skip" tabindex="-1"><g:message
code="default.link.skip.Label" default="Skip to content&hellip;" /></a>
        <div class="nav" role="navigation">
            <ul>
                <li><a class="home" href="${createLink(uri:
'/')}"><g:message code="default.home.Label" /></a></li>
                <li><g:link class="list" action="list"><g:message
code="default.list.Label" args="[entityName]" /></g:link></li>
                <li><g:link class="create" action="create"><g:message
code="default.new.Label" args="[entityName]" /></g:link></li>
                <li><g:link class="search" action="search"><g:message
code="Pesquisar Documento" args="[entityName]" /></g:link></li>
            </ul>
        </div>
        <div id="show-documento" class="content scaffold-show" role="main">
            <h1><g:message code="default.show.Label" args="[entityName]"
/></h1>
            <g:if test="${flash.message}">
            <div class="message" role="status">${flash.message}</div>
            </g:if>
            <ol class="property-list documento">

                <g:if test="${documentoInstance?.autor}">
                <li class="fieldcontain">
                    <span id="autor-label" class="property-
Label"><g:message code="documento.autor.Label" default="Autor" /></span>

                    <span class="property-value" aria-
labelledby="autor-label"><g:fieldValue bean="${documentoInstance}"
field="autor" /></span>

                </li>
                </g:if>

                <g:if test="${documentoInstance?.caminhoServidor}">
                <li class="fieldcontain">
                    <span id="caminhoServidor-label" class="property-
Label"><g:message code="documento.caminhoServidor.Label" default="Caminho Servi-
dor" /></span>
```

```

        <span class="property-value" aria-
labelledby="caminhoServidor-Label"><g:fieldValue bean="${documentoInstance}"
field="caminhoServidor"/></span>

    </li>
</g:if>

    <g:if test="${documentoInstance?.dataAtualizacao}">
    <li class="fieldcontain">
        <span id="dataAtualizacao-Label" class="property-
Label"><g:message code="documento.dataAtualizacao.Label" default="Data Atualiza-
cao" /></span>

        <span class="property-value" aria-
labelledby="dataAtualizacao-Label"><g:formatDate
date="${documentoInstance?.dataAtualizacao}" /></span>

    </li>
</g:if>

    <g:if test="${documentoInstance?.dataCriacao}">
    <li class="fieldcontain">
        <span id="dataCriacao-Label" class="property-
Label"><g:message code="documento.dataCriacao.Label" default="Data Criacao"
/></span>

        <span class="property-value" aria-
labelledby="dataCriacao-Label"><g:formatDate
date="${documentoInstance?.dataCriacao}" /></span>

    </li>
</g:if>

    <g:if test="${documentoInstance?.nomeArquivoOriginal}">
    <li class="fieldcontain">
        <span id="nomeArquivoOriginal-Label"
class="property-Label"><g:message code="documento.nomeArquivoOriginal.Label" de-
fault="Nome Arquivo Original" /></span>

        <span class="property-value" aria-
labelledby="nomeArquivoOriginal-Label"><g:fieldValue bean="${documentoInstance}"
field="nomeArquivoOriginal"/></span>

    </li>
</g:if>

    <g:if test="${documentoInstance?.paragrafos}">
    <li class="fieldcontain">
        <span id="paragrafos-Label" class="property-
Label"><g:message code="documento.paragrafos.Label" default="Download" /></span>

        <g:link action="download"
id="${documentoInstance.id}">${documentoInstance.nomeArquivoOriginal}</g:link>

    </li>
</g:if>

    <g:if test="${documentoInstance?.tipoArquivo}">
    <li class="fieldcontain">

```

```

        <span id="tipoArquivo-label" class="property-
Label"><g:message code="documento.tipoArquivo.Label" default="Tipo Arquivo"
/></span>

        <span class="property-value" aria-
labelledby="tipoArquivo-label"><g:fieldValue bean="${documentoInstance}"
field="tipoArquivo"/></span>

    </li>
</g:if>

</ol>
<g:form>
    <fieldset class="buttons">
        <g:hiddenField name="id" val-
ue="${documentoInstance?.id}" />
        <g:link class="edit" action="edit"
id="${documentoInstance?.id}"><g:message code="default.button.edit.Label" de-
fault="Edit" /></g:link>
        <g:actionSubmit class="delete" action="delete"
value="${message(code: 'default.button.delete.label', default: 'Delete')}" on-
click="return confirm('${message(code: 'default.button.delete.confirm.message',
default: 'Are you sure?')}');"/>
    </fieldset>
</g:form>
</div>
</body>
</html>

```

Search

A action Search é responsável por realizar a busca, por meio da *action search*, dos documentos por nome do autor, do arquivo ou palavras-chave.

```

<%@ page import="br.edu.unibalsas.Documento" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
<meta name="Layout" content="main"/>
<title> Busca </title>
</head>
<body>
    <div class="body">
        <g:form action="search" method="GET">
            <label for="tipoArquivo">
                <g:message default="Buscar" />
            </label>
            <g:textField name="q" value="${q}"/>
            <g:submitButton name="search" value="Buscar"/>
        </g:form>

    </div>

    <div id="list-documento" class="content scaffold-list" role="main">

```

```

<h1><g:message code="default.list.label" args="[entityName]"
/></h1>

<g:if test="${flash.message}">
<div class="message" role="status">${flash.message}</div>
</g:if>
<table>
    <thead>
        <tr>

            <g:sortableColumn property="autor" ti-
title="${message(code: 'documento.autor.label', default: 'Autor')}" />

            <g:sortableColumn proper-
ty="nomeArquivoOriginal" title="${message(code: 'documen-
to.nomeArquivoOriginal.label', default: 'Nome do Arquivo')}" />

            <g:sortableColumn proper-
ty="dataAtualizacao" title="${message(code: 'documento.dataAtualizacao.label', de-
fault: 'Data Atualizacao')}" />

            <g:sortableColumn property="dataCriacao"
title="${message(code: 'documento.dataCriacao.label', default: 'Data Criacao')}"
/>

            <g:sortableColumn property="tipoArquivo"
title="${message(code: 'documento.tipoArquivo.label', default: 'Tipo Arquivo')}"
/>

            <g:sortableColumn property="tipoArquivo"
title="${message(code: 'documento.tipoArquivo.label', default: 'Download')}" />

        </tr>
    </thead>
    <tbody>
        <g:each in="${documentoInstanceList}" status="i"
var="documentoInstance">
            <tr class="${(i % 2) == 0 ? 'even' : 'odd'}">

                <td><g:link action="show"
id="${documentoInstance.id}">${fieldValue(bean: documentoInstance, field: "au-
tor")}</g:link></td>

                <td>${fieldValue(bean: documentoInstance,
field: "nomeArquivoOriginal")}</td>

                <td><g:formatDate
date="${documentoInstance.dataAtualizacao}" /></td>

                <td><g:formatDate
date="${documentoInstance.dataCriacao}" /></td>

                <td>${fieldValue(bean: documentoInstance,
field: "tipoArquivo")}</td>

                <td><g:link action="download"
id="${documentoInstance.id}">${documentoInstance.nomeArquivoOriginal}</g:link></td>
            </tr>
        </g:each>

```

```

        </tbody>
    </table>
    <div class="pagination">
        <g:paginate total="${documentoInstanceTotal}" />
    </div>
</div>

</div>

</body>
</html>

```

UploadForm

Esta view disponibiliza por meio da *action uploadForm* a página para fazer o upload dos arquivos para a aplicação.

```

<%@ page import="br.edu.unibalsas.Documento" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
<meta name="layout" content="main"/>
<title> Upload File </title>
</head>
<body>
    <div class="body">
        <g:uploadForm action="upload" method="POST">
            <input type="file" name="myFile"/>
            <input type="submit"/>
        </g:uploadForm>
    </div>
</body>
</html>

```