UNIVERSIDADE FEDERAL FLUMINENSE
PRISCILA CALDAS PEREIRA

CONSTRUÇÃO DE UMA APLICAÇÃO PARA REPOSITÓRIOS ACADÊMICOS

Niterói
2016

PRISCILA CALDAS PEREIRA

CONSTRUÇÃO DE UMA APLICAÇÃO PARA REPOSITÓRIOS ACADÊMICOS

Trabalho de Conclusão de Curso submetido ao Curso de Tecnologia em Sistemas de Computação da Universidade Federal Fluminense como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Computação.

Orientador:
Leandro Soares de Sousa

NITERÓI
2016

PRISCILA CALDAS PEREIRA

CONSTRUÇÃO DE UMA APLICAÇÃO PARA REPOSITÓRIO ACADÊMICOS

Trabalho de Conclusão de Curso submetido ao Curso de Tecnologia em Sistemas de Computação da Universidade Federal Fluminense como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Computação.

Niterói, 15 de dezembro de 2016.

Banca Examinadora:

_____

Prof. Leandro Soares de Sousa, D.Sc. – Orientador

UFF – Universidade Federal Fluminense

_____

Prof. ou Profª. <NOME>, <Título>. – Orientador ou Avaliador

<Sigla da Universidade> - <Nome da Universidade>

Ficha catalográfica

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

P436 Pereira, Priscila Caldas
Construção de uma aplicação para repositórios acadêmicos / Priscila Caldas Pereira. – Niterói, RJ : [s.n.], 2016.
38 f.

Projeto Final (Tecnólogo em Sistemas de Computação) – Universidade Federal Fluminense, 2016.
Orientador: Leandro Soares de Sousa.

1. Desenvolvimento de software. 2. Repositório institucional. 3. Trabalho acadêmico. I. Título.

CDD 005.71

Este trabalho é dedicado a toda minha família, em especial aos meus pais e meus irmãos pelo suporte em todos os momentos, aos meus amigos pelo convívio durante toda essa jornada e ao meu marido Hugo por estar sempre ao meu lado me apoiando.

AGRADECIMENTOS

A Deus, que sempre iluminou a minha caminhada.

A meu Orientador Leandro Soares de Sousa pelo estímulo e atenção que me concedeu durante o curso.

Aos Colegas de curso pelo incentivo e troca de experiências.

A todos os meus familiares e amigos pelo apoio e colaboração.

RESUMO

Com o aumento da produção acadêmica se torna cada vez mais importante para uma Universidade ter um repositório acadêmico organizado e disponível para o maior número de pesquisadores possíveis, através da integração com diversos sistemas e outros repositórios do mundo. A falta de acesso a um trabalho científico já existente pode levar ao mal-uso do dinheiro, visto que o financiamento da pesquisa poderia ser empregado na continuidade de um trabalho existente, ao invés de refazê-lo para, muitas vezes, chegar num resultado semelhante.

Tendo em vista esse cenário, foi proposta uma solução em *software* que lida de forma simples e eficiente com o armazenamento e pesquisa de artigos dentro da Universidade.

Este software implementa o protocolo OAI-PMH e utiliza banco de dados MongoDB para que seja possível realizar pesquisas de forma eficiente.

Palavras-chave: MongoDB, WebServices, Ruby on Rails, NoSQL, OAI-PMH, Heroku.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

DP – *Data Providers* (Provedores de Dados)

LDB – Lei de Diretrizes e Bases

OAI – *Open Archives Initiative* (Inicitiva de Arquivos Abertos)

SP – *Service Providers* (Provedores de Serviços)

SUMÁRIO

1. INTRODUÇÃO

As pesquisas acadêmicas aumentam a cada dia e, dessa forma, muitos trabalhos acabam sendo replicados sem necessidade, atrasando pesquisas que poderiam ter uma continuidade, se fossem disponibilizadas, ao invés de uma replicação com abordagens semelhantes.

Muitos projetos acadêmicos começam através da pesquisa de assuntos relacionados que sejam relevantes, de forma que auxiliem o pesquisador no desenvolvimento de sua pesquisa, podendo identificar os problemas ainda em aberto, os resolvidos e os trabalhos que ainda podem evoluir.

Segundo a revista científica *Nature* [10], o gasto brasileiro com ciência é muito pouco eficiente. O país figura entre 50º entre os 53 países avaliados, ficando atrás de países como Irã, Paquistão e Ucrânia, e na frente do Egito, Turquia e Malásia. O *ranking* é elaborado através da divisão do número de artigos publicados em 68 revistas científicas internacionais de alto prestígio pelo total de investimentos em pesquisa. O ranking é liderado pela Academia Chinesa de Ciências, seguida por Harvard (EUA) e pela Sociedade Max Planck (Alemanha). A universidade latino-americana mais bem colocada é a USP, ela aparece em 271º lugar na “Nature”, seguida por UFRJ (557º), UNESP (574º) e Unicamp (613º).

Também de acordo com a mesma revista, o investimento anual do Brasil com pesquisa científica é de R$ 59,4 bilhões, somando as iniciativas públicas e privadas e o Brasil apesar de ser único país de seu continente que destina mais de 1% de seu PIB em pesquisa de desenvolvimento, não recebe muitas citações (quando outros artigos fazem referência a um determinado estudo), que são uma forma de medir um impacto de sua pesquisa no mundo.

Diante desse cenário, o presente trabalho acredita que apesar do alto investimento em pesquisa o número de repositórios acadêmicos ainda é baixo quando comparado com os países mais eficientes na pesquisa. Segundo estatísticas do

OpenDOAR (Directory of Open Access Repositories) [11], existem 91 repositórios no Brasil, enquanto que nos Estados Unidos 493 e na Alemanha 193.

Visto a importância de importância de repositórios para a eficiência nas pesquisas acadêmicas, este trabalho propõe um novo repositório capaz de se integrar com outros repositórios de documentos científicos, totalmente gratuito e *open-source,* que implementa o protocolo OAI-PMH através do modelo de *software* como serviço, disponível *online*, com capacidade de registro de provedores de dados e busca eficiente através da indexação de documentos.

O trabalho foi organizado na seguinte forma:

- O Capítulo 2 fará uma revisão da literatura, explicando um pouco sobre o que é um repositório acadêmico, sua história e importância. Além disso, abordará o protocolo OAI-PMH e o padrão de metadados *Dublin Core*, que são importantes para a implantação de um repositório.
- O Capítulo 3 introduzirá os conceitos da solução. Principais tecnologias utilizadas e suas importâncias para a qualidade da aplicação. Tecnologias como Ruby on Rails, Banco de dados NoSQL (MongoDB), *WebServices* e outras serão abordadas neste capítulo.
- O Capitulo 4 abordará a arquitetura proposta, explicando e exemplificando os componentes desenvolvidos e utilizados.
- O Capítulo 5 apresenta a conclusão do trabalho, contendo a indicação para trabalhos futuros.

2. REVISÃO DA LITERATURA

Este capítulo trata de uma revisão da literatura sobre a implementação repositórios institucionais e um estudo sobre as soluções atuais.

O conteúdo será apresentado nas próximas seções que serão divididas respectivamente em duas partes: a criação dos repositórios institucionais e a iniciativa *Open Archives* com o uso do protocolo OAI-PMH.

2.1 REPOSITÓRIOS INSTITUCIONAIS

Com o surgimento da Internet no final do século XX, a comunidade cientifica passou por grandes inovações na mídia eletrônica, o que trouxe profundas transformações e o início dos repositórios institucionais, que mudaram o modo como as instituições se comunicam e publicam seus resultados.

Uma definição para um repositório institucional seria: uma biblioteca digital destinada a guardar, preservar e garantir livre acesso, via internet, à produção científica no âmbito de uma dada instituição [1].

Repositórios também podem ser definidos como um “sistema de informação responsável por gerir e armazenar material digital” [2], e podem ser do tipo institucional, compreendendo a produção científica de uma instituição ou temáticos, abrangendo a produção científica de uma determinada área.

O aumento do uso de dispositivos com acesso a internet também tornou o acesso a informação possível para um número cada vez maior de pessoas, porém, ainda assim, o acesso a informação de qualidade como artigos científicos, que são insumos para o desenvolvimento tecnológico de uma população ainda é difícil. Muitos trabalhos são publicados por pesquisadores em revistas ou plataformas, que somente estão acessíveis mediante assinaturas, muitas destas de valor elevado para grande parte da população que precisa ter acesso ao conhecimento ali armazenado.

Com esse problema em pauta, propostas de um novo ambiente para armazenamento e busca de produções científicas vem sendo estudadas, e um exemplo é a iniciativa *Open Archives Initiative* (OAI) [3].

2.2 A INICIATIVA *OPEN ARCHIEVES* E O PROTOCOLO OAI-PMH

Segundo o Manifesto Brasileiro de Acesso Livre à Informação Científica, a iniciativa apresenta alguns princípios e padrões de interoperabilidade entre *softwares,* que levaram a criação de um paradigma de acesso livre à informação [3].

De acordo com o Instituto Brasileiro de Informação em Ciência e Tecnologia (IBICT), esta iniciativa também levou a criação de diversos *softwares* para construção de repositórios, como o *Open Journal Systems* (OJS), *DSpace* e outros. A criação de *softwares* como estes propiciam uma maior agilidade pela busca de artigos científicos, levando a trabalhos acadêmicos mais ricos (devido a maior quantidade de fontes para consulta e comparação de resultados) e que podem ser produzidos em um espaço de tempo menor, aumentando a produtividade do país na área acadêmica e, como consequência, contribuindo para melhor posicionamento no *ranking* mundial de produção acadêmica, além de possibilitar maior visibilidade dos pesquisadores em suas áreas de atuação. Em resumo, promove um melhor e mais rápido desenvolvimento da ciência.

Para que isso seja possível, não basta apenas a criação de padrões, mas também a adoção destes. Devido a isso foi necessário o estabelecimento de uma política nacional de acesso livre à informação científica, com apoio da comunidade científica. De acordo com o IBICT, os principais objetivos com essa iniciativa são a promoção do registro da produção científica brasileira, a disseminação da produção científica e o acesso livre à informação.

De acordo com Mueller [4], as pesquisas são normalmente realizadas por pesquisadores que trabalham em universidades públicas, com pesquisas que, em sua maioria, são financiadas pelo governo. A citação abaixo esclarece o modelo atual e a importância de um trabalho intenso para democratização do conhecimento gerado:

> "[...] o Estado [...] financia a educação dos novos cientistas, desde seu início até a obtenção dos graus mais altos [...]. Uma vez formado e já pesquisando, normalmente em uma universidade também mantida pelo Estado, sua pesquisa é frequentemente financiada pelas agências de fomento federais ou estaduais, vale dizer, de novo, dinheiro público. Terminada a pesquisa, sua divulgação em reuniões e congressos será de novo financiada pelo Estado. Finalmente, a publicação em revista indexada poderá também receber auxílios dos cofres públicos, pois em algumas áreas as editoras cobram dos autores por página publicada. Ao publicar em uma revista, é hábito o autor ceder às editoras o direito autoral sobre o artigo. (MUELLER, 2006, p. 33)."

Para viabilizar o uso da iniciativa OAI, o protocolo OAI-PMH (*Open Archives Initiative Protocol for Metadata Harvesting*) foi criado com objetivo de proporcionar a interoperabilidade entre bibliotecas/repositórios digitais. De acordo com o artigo "O Protocolo OAI-PMH para interoperabilidade em bibliotecas digitais" [5] e a própria especificação do protocolo disponível no site que o define [6] o protocolo é composto por um conjunto de seis verbos ou serviços que são invocados por HTTP que tornam possível a integração entre diferentes sistemas através da exposição dos metadados. Os metadados do protocolo seguem o padrão Dublin Core [7], um esquema que tem como objetivo descrever arquivos digitais através de 15 elementos, sendo alguns deles o título do artigo, os autores, resumo, palavras-chave, data, formatos, idioma, entre outros. Esses metadados são exemplificados na figura 1.

Figura 1 – *Exemplo de descrição bibliográfica em formato Dublin Core* [5]

O protocolo OAI-PMH, com sua última versão (2.0) implementada em 2002, introduz o conceito de *Metadata Harvesting*, onde os provedores de serviços acessam os repositórios cadastrados na OAI e realizam buscas através dos metadados disponibilizados. Esse protocolo torna mais fácil a integração entre os diferentes sistemas de repositórios digitais. A figura 2 ilustra a comunicação entre os provedores que compõe a solução:

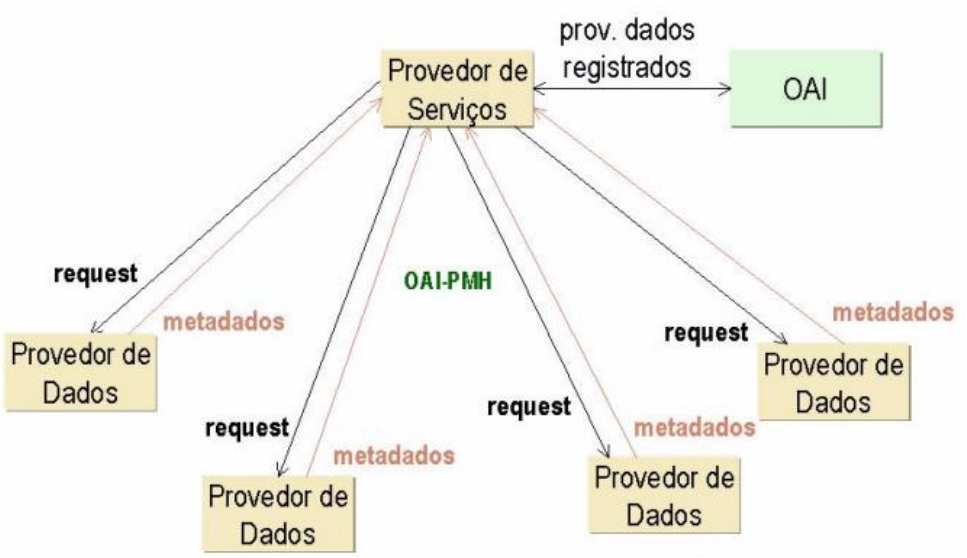

Figura 2 – *Metadata Harvesting* [5]

Os verbos mais utilizados pelo protocolo são:

- Identify - Utilizado para recuperação de informações do provedor de dados (nome, url, e-mail do administrador, etc).
- ListMetadataFormats – Retorna a lista de todos os metadados disponíveis no repositório ou para de um item específico
- GetRecord – Retorna um metadado específico e único do repositório
- ListRecords – Retorna os registros de um repositório, podendo receber argumentos opcionais para uma seleção dos registros a serem retornados.

Com base nos trabalhos relacionados, este trabalho tem como objetivo a criação de um repositório de documentos científicos totalmente gratuito e *open-source* que implementa o protocolo OAI-PMH através do modelo de *software* como serviço, disponível *online*, com capacidade de registro de provedores de dados e busca eficiente através da indexação de documentos.

3. DETALHAMENTO DA SOLUÇÃO

3.1 INTRODUÇÃO

Como proposta de solução foi desenvolvido um *software* para gerenciamento de repositórios acadêmicos com capacidade de indexação de documentos como artigos científicos essenciais para a comunidade acadêmica. Para isso, tecnologias como Banco de Dados NoSQL tem seu papel importante na arquitetura proposta, pois oferecem alta performance para busca e indexação dos dados.

Diante desse cenário o Banco de Dados MongoDB foi escolhido por ter uma arquitetura orientada a documentos e por consequência o *framework* Ruby on Rails também foi escolhido por ter uma excelente integração com este Banco de Dados, apresentar uma comunidade ativa e possibilitar o desenvolvimento de aplicações *web* de forma simples e rápida.

Uma explicação mais detalhada sobre o MongoDB e sua arquitetura será apresentada neste capítulo, assim como a comparação de seu uso com outros Bancos de Dados SQL. São exemplos de Bancos de Dados SQL: PostgreSQL, MariaDB. MySQL.

Neste capitulo também será apresentado o framework Ruby on Rails, seus principais conceitos, utilizações e integração com o MongoDB.

Por último, uma proposta de modelagem será discutida levando em consideração as diferenças de arquitetura impostas por Bancos de Dados NoSQL e a solução implementada será apresentada.

3.2 TECNOLOGIAS UTILIZADAS

3.2.1 BANCO DE DADOS NOSQL

A cada ano mais e mais dados estão sendo gerados no mundo e auxiliando grande tomadas de decisões. Um exemplo desse crescimento pode ser visto numa frase dita pela empresa IBM: em 2013, 90% dos dados do mundo tinham sido criados nos últimos 2 anos.

Essa frase mostra um crescimento exponencial dos dados no mundo e com isso, os bancos relacionais começaram a apresentar alguns problemas como por exemplo, pouca agilidade para mudanças na estrutura das tabelas, dificuldade para distribuição e processamento de grande volume de dados de forma rápida e eficaz.
Para suprir esses problemas, o termo NoSQL (Não Somente SQL) surgiu e em 1998, como um nome para um banco de dados relacional de código aberto que não possuía uma interface SQL (Linguagem de Consulta Estruturada). Seu autor, Carlo Strozzi alega que o movimento NoSQL “é completamente distinto do modelo relacional e, portanto, deveria ser mais apropriadamente chamado ‘NoREL’ ou algo que produzisse o mesmo efeito”. Porém, o termo só voltou a ser assunto em 2009, por um funcionário do Rackspace, Eric Evans, quando Johan Oskarsson, da Last.fm, queria organizar um evento para discutir os bancos de dados open source distribuídos [8].

NoSQL não é uma campanha contra a linguagem SQL e sim um novo tipo de sistema de armazenamento que surgiu para suprir necessidades que os bancos de dados tradicionais (relacionais) não estavam conseguindo atender. Algumas características desses sistemas são: alta escalabilidade, baixo custo, facilidade para distribuição, flexibilidade no esquema.

Grande parte dos bancos NoSQL possuem toda a informação necessária agrupada no mesmo registro, ou seja, não existem relacionamentos entre várias tabelas para formar uma informação, ela estará em sua totalidade no mesmo registro. Um exemplo disso, é que a maioria desses bancos não dão suporte a *JOIN* de tabelas.

Os Bancos NoSQL podem ser subdivididos pela forma como trabalham com dados. Alguns exemplos são: Bases de Dados Chave-Valor (Key/ Value Store): esse é o tipo de banco de dados NoSQL mais simples. E, por uma boa razão, embora ofereça um conjunto limitado de recursos, possui uma característica importante: alto desempenho. O conceito principal é uma chave e um valor para essa chave. Alguns exemplos são: Berkeley DB, gdbm/ndbm, Tokyo Cabinet, Hamsterdb, Performance numbers, Project Voldermort, MemcacheDB, SimpleDB;

1) Colunas (Wide Columns Store): Diferentemente do exemplo assim, neste caso, existem tabelas que contêm colunas. A diferença é que neste caso existe uma abordagem hibrida de características declarativas de bancos de dados relacionais com o para chave-valor e esquema de variáveis armazenadas como chave-valor. Esse tipo de banco armazrna tabelas de dados como seções de colunas ao invés de linhas de dados. Alguns exemplos são: HBase (Apache), HiperTable, Cassandra (Apache);

2) Grafos (Graph Store): É um tipo de banco de dados que utiliza estruturas de gráfico para consultas semânticas como nós, arestas e propriedades para representar e armazenar dados. Um conceito-chave do sistema é o gráfico (ou aresta ou relação), que relaciona diretamente dois itens do banco. As relações permitem que dois dados sejam ligados diretamente, e em muitos casos, recuperados com uma única operação. Com uma complexidade maior, esses bancos de dados guardam objetos, e não registros como os outros tipos de NoSQL. A busca desses itens é feita pela navegação desses objetos. Alguns exemplos são: HyperGraphDB, BigData , Neo4J, InfoGrid.

3) Orientado a colunas (Column Oriented Store). São bancos orientados a coluna, esta forma de armazenamento tem como principal vantagem armazenar os dados em colunas, ajudando na escalabilidade e diminuindo a quantidade de I/O, que é a entrada/ saída de dados durante as operações. São bancos de dados relacionais, porém

apresentam características do NoSQL. Alguns exemplos são: MonetDB, Vertica, LucidDB, Infobright, Ingres/ Vectorwise.

4) Documentos (Document Store): Um banco de dados orientado a documentos é projetado para armazenar, recuperar e gerenciar dados orientados a documentos ou semi-estruturados. Esses bancos assumem que os documentos encapsulam e modificam dados (ou informações) em algum padrão (ou codificação). As codificações em uso incluem XML, YAML, JSON, BSON, bem como formulários binários como PDF e documentos do Microsoft Office. Alguns exemplos são: CouchDB (Apache), Riak, RavenDB, MongoDB. Este último foi escolhido para o presente trabalho e será melhor abordado no próximo capítulo;

Como apresentado na figura 3, os bancos chave-valor aguentam maior quantidade de registros, enquanto os outros grafos são mais complexos.

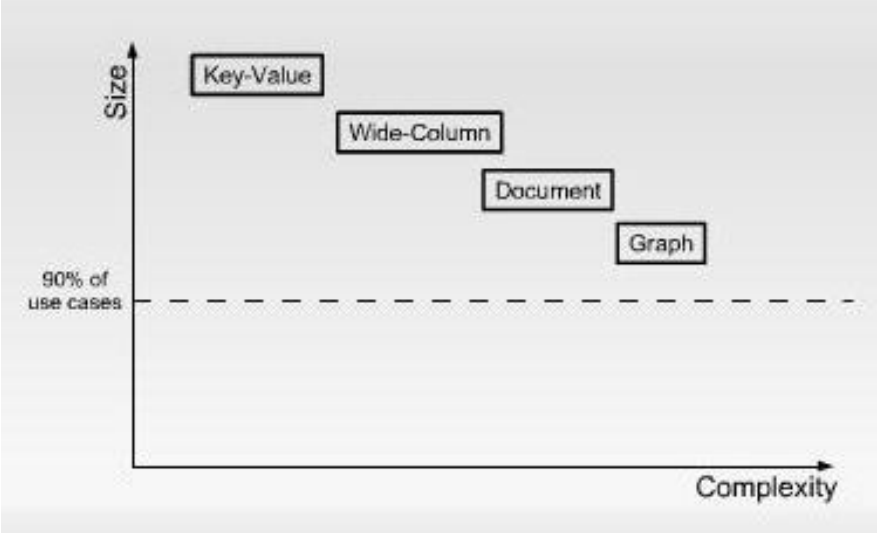

Figura 3 – Categorias de Bancos NoSQL [8]

Em um projeto, não necessariamente é preciso usar apenas banco de dados NoSQL, é possível usá-lo em conjunto com outro banco de dados. Os bancos NoSQL são indicados para grandes cargas de dados, exigência de velocidade na consulta e escrita em grandes volumes de dados.

Ainda segundo o grafo, o uso de banco de dados NoSQL é indicado para a apenas 10% das aplicações. Na maioria dos casos (90%), as aplicações podem usar um banco de dados relacional, que são definidos como um conjunto coletivo de vários conjuntos de dados organizados por tabelas, registros e colunas, que estabelecem uma relação bem definida entre tabelas de banco de dados e utilizam uma linguagem de consulta estruturada (SQL. Esses bancos baseiam-se em que todos os dados estão guardados em tabelas, pelo conceito de entidade e relacionamento. Os dados são separados de forma única tentando diminuir ao máximo a redundância, pois a informação é criada pelo conjunto dos dados, onde são as relações entre as tabelas que fazem esse serviço. As principais características são: tabelas, schema definido, hierarquia, redundância mínima, entidade e relacionamento, formas normais, transações, ACID (Atomicidade, Consistência, Isolamento, Durabilidade).

Neste trabalho optou-se por usar um banco de dados NoSQL, pois esse tipo de organização é uma solução alternativa para os bancos de dados relacionais, com alta escalabilidade e desempenho. O uso dessa tecnologia permite ao repositório um modelo de dados mais simples, com as seguintes características: escalabilidade horizontal, streaming, esquema menor, desenvolvimento rápido e flexibilidade para alteração no schema, camada de cache, variedade ampla de tipos de dados, objetos binários, carregamento em massa, tolerância à falha, escalabilidade, *clusterização, mapreduce, sharding*.

3.2.1.1 MongoDB

O MongoDB é um banco de dados orientado a documentos multi-plataforma e *open-source*, um tipo de banco de dados NoSQL. Como banco de dados NoSQL, o MongoDB evita a estrutura baseada em tabelas do banco de dados relaciona para adaptar documentos semelhantes a JSON que possuem esquemas dinâmicos que ele chama BSON.

Isso torna a integração de dados para certos tipos de aplicativos mais rápida e fácil. O MongoDB é construído para escalabilidade, alta disponibilidade e desempenho de uma implantação de servidor para grandes e complexas

infraestruturas multi-site. Foi desenvolvido primeiramente por MongoDB Inc., em outubro de 2007, originalmente como uma parte principal em um produto de PaaS (plataforma como um serviço) similar a Windows Azure e Google App Engine. O desenvolvimento tornou-se open source em 2009.

É um dos bancos de dados mais popular entre os bancos NoSQL, sendo usado como backend para muitos sites importantes, incluindo eBay, The New York Times. O MongoDB está disponível sob a GNU Affero General Public License enquanto seus drivers de idioma estão disponíveis sob a Licença Apache. Há também licenças comerciais sendo oferecido.

Algumas características do MongDB são:

- Consultas ad hoc: suporta pesquisas por campo, buscas de expressões regulares e consultas de intervalos;
- Indexação: qualquer campo do documento BSON pode ser indexado;
- Replicação - fornece alta disponibilidade através de conjuntos de réplicas que consiste em duas ou mais cópias dos dados originais.
- Balanceamento de carga - sharding é o método usado para permitir que o MongoDB seja escalado horizontalmente, o que significa que os dados serão distribuídos e divididos em intervalos e então armazenados em diferentes fragmentos que podem ser localizados em diferentes servidores. As chaves Shard são usadas para determinar como os dados serão distribuídos.
- Agregação - MapReduce pode ser aplicada para permitir o processamento em lote de dados, bem como executar operações de agregação.
- Armazenamento de arquivos - O MongoDB pode ser usado como sistema de arquivos que faz uso das funções acima e atua de forma distribuída através de sharding.

3.2.2 RUBY ON RAILS

O Ruby on Rails é um *framework* para desenvolvimento web *opensource* que vem sido amplamente utilizado nos últimos anos, ganhando destaque pela sua simplicidade e rapidez no desenvolvimento de *features*. Algumas grandes instituições, como a Universidade Federal Fluminense, já utilizam esse framework em grande parte de suas aplicações. Com a adoção do mesmo, foi possível um ganho de produtividade e confiabilidade muito grande que resultou em mais sistemas criados em menos tempo, gerando menor custo. A escolha pelo framework foi devido a essas facilidades e também pelo maior domínio já existente sobre a ferramenta. Exemplo de empresas que utilizam Ruby on Rails em produção são: Groupon, ESPN, NewRelic, Github, Shopify, Bloomberg, Airbnb, SoundCloud e outros.

O *framework* Rails pode ser visto como uma ferramenta para desenvolvimento web utilizando a linguagem de programação Ruby, assim como o *framework* web Django é utlizado para a linguagem Python ou o Spring MVC para o Java. Aplicações desenvolvidas em Ruby on Rails são divididas em camadas que são conhecidas como MVC – Model, View e Controller. Um esquema que ilustra essa arquitetura é representado na figura 4.

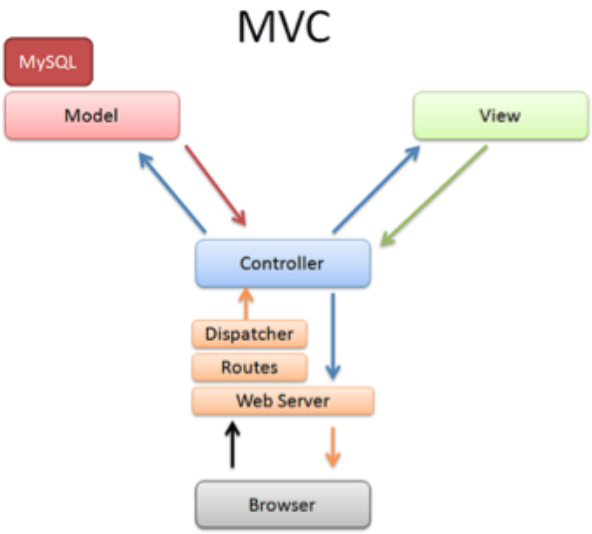

Figura 4 – Ruby on Rails - Arquitetura MVC [5]

Quando a requisição é feita pelo *browser*, ela passa pela camada de Web Server, onde podem ser usados as mais diversas ferramentas, porém sendo comumente utilizado em produção o Apache ou o Nginx. Quando a camada de Web Server recebe a requisição, ela encaminha para o módulo do Rails responsável pelo roteamento (*ActionDispatch::Routing*) que verifica qual endereço está sendo requisitado, encaminhando então para o *controller* correspondente de acordo com um arquivo pré-definido (routes.rb) que contém todas as rotas cadastradas pelo desenvolvedor.

A camada de *controller* é responsável por receber os parâmetros da requisição que foram encaminhados pelo *routes* e produzir o output apropriado. Para isso, muitas vezes necessária a comunicação com as outras camadas: *Model* e *View*. A camada de modelo contém regras de negócio, regras de consistência em banco de dados (como por exemplo, quais colunas são obrigatórias) e é responsável por interagir diretamente com o banco de dados, tendo a capacidade de salvar, apagar, buscar ou inserir dados no mesmo.

A camada de *View* é responsável por apresentar ao usuário a saída que é vista comumente em HTML. Ela também tem a capacidade de se comunicar com outras camadas, como o *model* e o *controller*. Isso acontece quando é necessário exibir na tela informações contidas em banco de dados, como por exemplo o nome e matrícula dos alunos de Tecnologia de Sistemas de Computação do Cederj, ou até mesmo mensagens personalizadas indicando o sucesso de uma determinada ação. O ideal é que não seja inserida lógica de programação diretamente na *View* para que se tenha baixo acoplamento, conceito visto em Engenharia de Software.

Como resumo da arquitetura em funcionamento, o cenário de um aluno efetuando seu *login* no sistema acadêmico da UFF e como essa requisição interage em meio as camadas do MVC é apresentado abaixo.

Na ordem, o que ocorre é:

1. O aluno acessa um endereço, como por exemplo https://sistemas.uff.br/iduff

2. O DNS é resolvido para um IP, que contém um Web Server sendo executado em determinada porta

3. A requisição é direcionada para esse Web Server

4. O Web Server direcionada a requisição para o *ActionDispatch* do Rails

5. O *ActionDispatch* compara o endereço pedido com as rotas previamente escritas no arquivo routes.rb e envia para o *Controller* responsável por receber aquela requisição

6. Caso a requisição tenha sido feita para uma página de index, o *controller* irá se comunicar com a View e solicitar que a página seja *renderizada*

7. A requisição volta para o browser do aluno que agora pode navegar normalmente.

Caso o aluno efetue *login*, por exemplo, o *controller* também fará comunicação com o *model* para que os campos preenchidos com CPF e senha possam ser validados no banco de dados. Sendo a operação efetuada com sucesso ou não, a *view* deverá ser capaz de retornar uma mensagem como "*Login* efetuado com sucesso" ou "CPF e/ou senha inválidos".

Essa arquitetura apresentada acima foi utilizada em todo o software, que será explicado com mais detalhes nas seções adiante.

3.2.2.1 Ruby Gems

Como a maioria das linguagens de programação, Ruby utiliza um grande conjunto de bibliotecas de terceiros. A maioria delas são disponibilizadas na forma de uma gem. RubyGems é um sistema de pacotes Ruby que facilita a criação, compartilhamento e instalação de bibliotecas (pode ser exemplificado como um sistema de distribuição similar ao apt-get do Linux, porém voltado a *software* em Ruby).

O processo de instalação pode variar, geralmente há um arquivo README

ou INSTALL com instruções sobre isso.

O principal lugar onde as bibliotecas são hospedas é o RubyGems.org, que disponibiliza bibliotecas Ruby como gems.

Projetos Ruby on Rails possuem um arquivo Gemfile onde o desenvolvedor pode especificar as gems que pretende usar e suas verões. Um exemplo de Gemfile, que foi utilizado neste projeto pode ser visto na figura 5.

```
source 'https://rubygems.org'

gem 'rails', '4.2.7.1'
gem 'sass-rails', '~> 5.0'
gem 'uglifier', '>= 1.3.0'
gem 'coffee-rails', '~> 4.1.0'
gem 'jquery-rails'
gem 'turbolinks'
gem 'bootstrap_sb_admin_base_v2'
gem 'mongoid'
gem 'bson_ext'
gem 'devise'
gem 'mongoid-paperclip'
gem 'has_scope'
gem 'oai'
```

Figura 5 – Arquivo de Gems do Projeto

Algumas gems foram essenciais para a execução deste projeto, pois permitiram a integração do framework com o MongoDB, a inclusão de arquivos anexos, uma identidade visual, autenticação de usuários e o uso do protocolo OAI-PMH. Segue abaixo uma breve explicação sobre essas bibliotecas:

1. Mongoid - https://rubygems.org/gems/mongoid
   Biblioteca foi utilizada para fazer a integração do MongoDB com o Ruby on Rails. É a estrutura oficialmente suportada do ODM (*Object-Document-Mapper*) para o MongoDB em Ruby.

2. Mongoid-paperclip - https://rubygems.org/gems/mongoid-paperclip
   Biblioteca utilizada para o anexar os arquivos. Sua configuração permite tratar os arquivos de forma muito semelhante a outros atributos e gerenciar validações com base no tamanho e presença, se necessário. Pode transformar o anexo de tamanho, se necessário, e o principal pré-requisito d instalação é o ImageMagick. Arquivos anexados são salvos no sistema de arquivos e referenciados no navegador por uma especificação facilmente compreensível, que tem padrões válidos e úteis.

3. Bootstrap_sb_admin_base_v2
   https://rubygems.org/gems/bootstrap_sb_admin_base_v2
   É uma coleção de vários elementos e funções personalizáveis para projetos da web, empacotados previamente em uma única ferramenta. Ao projetar um site com o Bootstrap, os desenvolvedores podem escolher quais elementos querem usar. E, o mais importante, podem ter a certeza de que os elementos escolhidos não conflitarão entre si. Esses elementos personalizáveis contidos no Bootstrap são uma combinação de HTML, CSS e JavaScript.
   A identidade visual da aplicação foi feita utilizando o tema Bootstrap_sb_admin_base_v2, que pode ser encontrado no site: https://startbootstrap.com/

4. Devise - https://rubygems.org/gems/devise
   Biblioteca utilizada para implementar facilmente a autenticação dos usuários na aplicação. Algumas características fundamentais são:
   1. Baseado em rack;
   2. Solução MVC completa baseada em Rails;
   3. Permite ter vários modelos conectados ao mesmo tempo
   4. Baseado no conceito de modularidade: use apenas o que precisa;
   5. Composto por um conjunto de 10 módulos: Banco de dados Authenticatable, Omniauthable, Confirmável, Recuperável, Registável, Lembrável, Rastreável, Timeoutable, Validatable, Bloqueado;

5. Oai - https://rubygems.org/gems/oai

Biblioteca utilizada para implementar o *Open Files Protocol para Metadata Harvesting (OAI-PMH)*. O OAI-PMH é um protocolo utilizado para compartilhar metadados entre repositórios de bibliotecas digitais. A especificação OAI-PMH define seis verbos (Identify, ListIdentifiers, ListRecords, GetRecords, ListSets, ListMetadataFormat) usados para descoberta e compartilhamento de metadados. A gem ruby-oai inclui uma biblioteca de cliente, uma biblioteca de servidor / provedor e um shell de coleta interativo.

3.2.3 WEB SERVICES

WebServices são comumente utilizados para que seja possível a interoperabilidade entre sistemas. Isso quer dizer que através do uso deste, é possível que duas ou mais aplicações se comuniquem e troquem informações através do protocolo HTTP. Uma grande vantagem é que a troca de mensagens entre as aplicações é feita através de um padrão geralmente apresentado nos formatos XML, JSON ou SOAP. Sendo assim, é possível que aplicações escritas em linguagens distintas consigam se comunicar.

Quando se tem um ambiente onde as aplicações se comunicam através dos WebServices, há uma maior liberdade de resolver problemas computacionais usando a linguagem que melhor convir para a solução do mesmo. Esse conceito foi fortemente empregado no *software* proposto neste trabalho.

Imagine se fosse necessário consultar um determinado atributo de uma pessoa (data de nascimento), que apenas outra aplicação tem acesso direto (via banco de dados), e então fosse necessário liberar acesso para a consulta desse atributo em determinada tabela do banco de dados para um determinado IP (onde provavelmente a aplicação que deseja consumir o recurso está hospedada) apenas para que essa consulta possa ocorrer.

Se a aplicação precisa de uma informação que está contida de forma espalhada em diversos bancos de dados diferentes esse cenário é ainda mais crítico. Imagine ainda um cenário em que a UFF necessita de acesso a dados que apenas o Ministério da Educação é capaz de fornecer em tempo real. Para que isso seja possível, sem o uso de WebServices, você deveria solicitar acesso a todos esses

bancos de dados. Um cenário de caos generalizado é então configurado, sendo impossível de manter e garantir segurança a médio/longo prazo.

Para a utilização de um WebService, é necessário que o responsável pela implementação deste crie uma documentação confiável que represente de forma fiel o retorno dos dados para uma certa entrada. É comum que seja exigido, caso o serviço não seja público, uma chave de identificação do requisitante para que o serviço seja atendido. Um detalhe importante é que, caso as informações trafegadas sejam privadas ou contenham dados sensíveis, deve-se utilizar o protocolo SSL para que as requisições não sejam interceptadas e capturadas em texto plano por ataques do tipo *man-in-the-middle*.

Espera-se também, como boa prática, que os devidos status HTTP sejam retornados corretamente. Caso os dados sejam retornados com sucesso, o status retornado deverá ser 200 OK, caso ocorra alguma falha de autenticação, o retorno esperado recebe o código 401, caso o item que você esteja buscando (geralmente enviado como parâmetro) não seja encontrado, o retorno deve conter o código 404, e assim por diante. Esses códigos, quando utilizados de forma correta ajudam o desenvolvedor a tratar possíveis problemas sem depender de implementações do servidor, como por exemplo a comparação de uma determinada palavra, que causaria grande acoplamento, implicando em grande risco de falha na integração caso essa palavra seja alterada pelo desenvolvedor do WebService.

4. ARQUITETURA E SOLUÇÃO

Conforme visto na seção anterior, o *software* desenvolvido utiliza WebServices para comunicação com outros repositórios acadêmicos através do protocolo OAI-PMH, permitindo assim não apenas buscas no repositório local, mas também em repositórios remotos, aumentando a chance de um pesquisador encontrar o material procurado. Os componentes utilizados para o desenvolvimento da solução proposta são divididos de forma desacoplada nas camadas de Banco de Dados, APIs e *backend*. Sendo assim, para cumprir os requisitos do projeto, as seguintes tecnologias foram adotadas no desenvolvimento:

- Ruby on Rails como *framework web,* devido ao fato ser uma tecnologia de fácil aprendizado, fácil *deployment* através de ferramentas como o Heroku, e conter uma grande comunidade de desenvolvedores que mantém o projeto.

- Banco de dados MongoDB, pelo fato de ser um banco de dados *NoSQL* com suporte nativo a documentos, ter amplo uso no mercado de tecnologia, e apresentar resultados satisfatórios com relação a indexação e busca dos documentos nele armazenados. Sua capacidade de distribuir os dados em *cluster* também foi considerada, pois pode apresentar um diferencial no futuro, quando a quantidade de dados crescer consideravelmente.

A grande vantagem de se adotar arquiteturas como essa é a maior capacidade que o sistema adquire de adequar a diferentes demandas, podendo ser considerada então uma arquitetura escalável, requisito fundamental na Engenharia de Software para projetos de sistemas de alta disponibilidade e confiança.

A arquitetura proposta é exemplificada através da figura 6, onde é possível verificar as interações entre os diferentes componentes citados anteriormente.

Uma vez que o usuário entre na plataforma e procure determinado artigo pelos parâmetros desejados, o sistema consulta sua base de dados local para verificar se algum artigo corresponde aos parâmetros de entrada e, posteriormente, consulta as diversas bases de dados remotas que estão cadastradas a procura de artigos correspondentes, retornando assim um resultado possivelmente mais completo para o usuário, contendo um maior número de artigos. Para que isso ocorra, a comunicação entre o sistema e os diferentes repositórios deve ser feita através do protocolo OAI-PMH, que padroniza esse tipo de comunicação e já é amplamente utilizado.

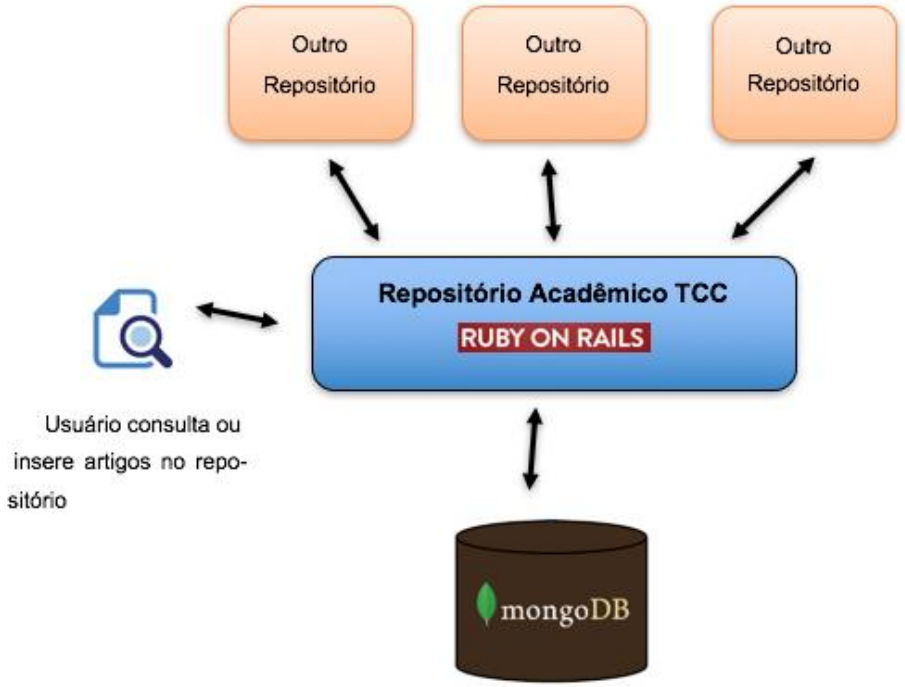

Figura 6: Arquitetura da solução

O MongoDB foi estruturado inicialmente com duas *collections*, similares as tabelas do banco de dados relacional. A primeira foi chamada de *Documents* e contém todos os metadados do protocolo Dublin Core para repositórios, e a segunda chamada *Users* que contém os dados dos usuários para controle de acesso ao sistema e informações relativas ao mesmo. As *collections* citadas estão exemplificadas nas figuras 7 e 8, respectivamente.

Figura 7: *Collection* Documents

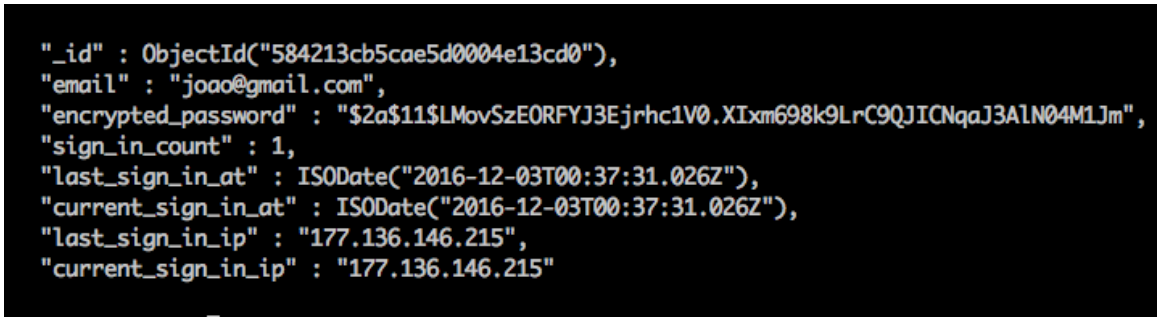

Figura 8: *Collection* Users

Como pode ser observado em ambas figuras, os dados são armazenados em uma estrutura flexível, conhecida como *schemaless*, comum em bancos de dados *NoSQL,* e é representada através de uma estrutura semelhante a um JSON.

As *collections Documents* e *Users* se relacionam através do atributo user, contido na *collection Documents*. Esse atributo foi criado com objetivo de tornar possível relacionar usuários e seus artigos acadêmicos.

Alguns dos atributos contidos na *collection* Users são responsáveis por manter o acesso ao sistema mais seguro, sendo possível verificar de onde estão sendo realizados os acessos, através do número de IP, a quantidade de acessos

realizados por um determinado usuário, e a gravação de sua senha de forma cifrada, fundamental para evitar o acesso indevido de um usuário caso esteja tenha acesso aos dados desta *collection*. Deve-se levar em consideração que muitos usuários repetem suas senhas em diversos *sites*, o que aumenta ainda mais a responsabilidade de quem desenvolve, visto que uma falha de segurança no sistema desenvolvido pode levar ao acesso indevido a outros também, havendo vazamento de senhas de usuários em determinados serviços pela internet.

A cifragem de senhas é um dos componentes importantes no desenvolvimento de um sistema seguro, visto que o uso de funções *hash* implementam uma cifra de mão única, o que quer dizer que dado uma entrada, a saída é sempre a mesma (o *hash*), porém, dado um *hash*, não é possível descobrir sem métodos de força bruta qual é a entrada que o originou.

Para aumentar ainda mais o nível de segurança da senha do usuário, pode-se utilizar diversos algoritmos de *hash*, sendo comum o uso do md5, sha1, sha256 e bcrypt, cada um deles implementado com uma quantidade de bits diferente, ou seja, quanto maior o número de bits utilizado, mais segura a senha estará. Um cuidado que deve ser ponderado é que não basta escolher sempre o maior número de bits possível, visto que a cifragem de uma entrada utilizando muitos bits pode consumir alto recurso computacional, podendo refletir em lentidões na página para o usuário final. Levando em consideração os fatos apresentados, o algoritmo *bcrypt* [12] foi escolhido para utilização por ser amplamente usado e apresentar boa performance, apesar de utilizar 448 bits para formação das cifras.

Uma vez que temos informações sobre os dados de acesso de um usuário, é possível personalizar a experiência deste no repositório através da criação de *links* para os seus artigos e recomendando novos artigos de acordo com suas preferências. A possibilidade de adaptação de um sistema de acordo com o uso de usuário permite que uma melhor interação do mesmo, levando a uma melhor experiência de uso.
A tela da página inicial do sistema é apresentada na figura 9.

Figura 9: Tela inicial da Aplicação

Conforme pode ser verificado na figura 9, os ícones coloridos representam os atalhos personalizados para o usuário que utiliza o sistema e, logo abaixo, os artigos do usuário João Manoel são listados, podendo o usuário efetuar uma edição com objetivo de adicionar novos metadados, ou até mesmo excluí-lo da base local. Os itens coloridos foram divididos da seguinte forma:

- Meus Artigos: possibilita um link direto do usuário aos artigos que publicou na plataforma;
- Artigos da UFF: possibilita que usuário veja todos os artigos publicados por sua Universidade. Neste exemplo, a UFF;
- Recomendados: possibilita que o usuário tenha acesso a artigos recomentados, que serão filtrados de acordo com as preferências que o usuário cadastrar na plataforma. Como por exemplo, áreas de interesse e autores;
- Artigos deste Ano: possibilita que o usuário tenha acesso a todos os artigos publicados no último ano;

Por fim, a aplicação foi disponibilizada no Heroku, um famoso PaaS (*Plataform as a Service*), que tem como objetivo retirar a complexidade de

manutenção do sistema para o usuário final no que diz respeito principalmente aos servidores. Dessa forma, até usuários com pouco conhecimento conseguem disponibilizar suas aplicações através de um endereço na internet, visto que a administração de servidores, infraestrutura, rede, e banco de dados é realizada pelo Heroku. A capacidade de escalar aplicações utilizando um conceito desta ferramenta chamado de Dynos, que são representações de recursos computacionais, também facilita a gerencia em casos onde a escalabilidade é importante, como por exemplo em momentos do ano em que a submissão de artigos é mais alta. Caso a aplicação necessite de mais performance, basta solicitar mais Dynos ao Heroku.

5. CONCLUSÕES E TRABALHOS FUTUROS

Através desse trabalho é possível concluir que o número de repositórios ainda é muito baixo, sendo o Brasil muito pouco eficiente em suas publicações acadêmicas. Desta forma, faz-se necessário a construção de mais repositórios que contribuam para projetos relevantes e este trabalho contribui oferecendo um repositório totalmente *open source* e de fácil instalação.

No entanto, para que iniciativas como estas funcionem e tragam os benefícios necessários para a comunidade acadêmica, não bastam só ferramentas e softwares, pois é preciso o engajamento de alunos, professores e pesquisadores. Isso torna possível a melhoria e aprofundamento de trabalhos já existentes e uma maior eficiência dos gastos brasileiros com pesquisas.

Espera-se com este trabalho que soluções com esta, ajudem o desenvolvimento da comunidade acadêmica e possam ser ampliadas através de novos serviços como comunicações sobre novas publicações e interação entre pesquisadores brasileiros.

Uma proposta para trabalho futuro seria a criação de um módulo para o sistema que utilizaria um banco de dados NoSQL baseado em grafos, como por exemplo o Neo4J, para a apresentar e realizar análises sobre as relações entre os artigos publicados, exibindo grupos de pesquisas comuns, identificação de coautores e temas relevantes levando em consideração as citações dos mesmos.

6. BIBLIOGRAFIA

1. Sayão, Luis; Toutain, Lídia Brandão; Rosa, Flavia Garcia; Marcondes, Carlos Henrique**. Implantação e gestão de repositórios institucionais: políticas, memória, livre acesso e preservação**. Editora: EDUFBA, 2009.
2. FERREIRA, MIGUEL. **Introdução à preservação Digital**. Editora: Escola de Engenharia da Universidade do Minho, 2006. Disponível em: <https://repositorium.sdum.uminho.pt/bitstream/1822/5820/1/livro.pdf >. Acesso em: 17 de agosto de 2016.
3. IBICT. **Manifesto Brasileiro de Apoio ao Acesso Livre à Informação Científica**. Disponível em: < http://livroaberto.ibict.br/Manifesto.pdf>. Acesso em: 19 de agosto de 2016.
4. Mueller, Suzana Pinheiro Machado. **A comunicação científica e o movimento de acesso livre ao conhecimento.** Disponível em: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0100-19652006000200004>. Acesso em: Acesso em: 2 de setembro de 2016.
5. Garcia, Patrícia de Andrade Bueno; Sunye, Marcos Sfair. **O Protocolo OAI-PMH para interoperabilidade em bibliotecas digitais**. Disponível em: <http://conged.deinfo.uepg.br/~iconged/Artigos/artigo_09.pdf>. Acesso: 2 de setembro de 2016
6. **The Open Archives Initiative Protocol for Metadata Harvesting**. Disponível em: <https://www.openarchives.org/OAI/openarchivesprotocol.html>. Acesso em: 5 de setembro de 2016.
7. **Metadata Dublin Core**. Disponível em: <http://dublincore.org/>. Acesso em: 5 de setembro de 2016.
8. **O que é NoSQL.** Disponível em: <http://imasters.com.br/artigo/17043/banco-de-dados/nosql-voce-realmente-sabe-do-que-estamos-falando>. Acesso em: 15 de outubro de 2016.
9. **Documentos NoSQL.** Disponível em: < https://docs.microsoft.com/pt-br/azure/documentdb/documentdb-nosql-vs-sql>. Acesso em: 16 de outubro de 2016.

10. **Gasto Brasileiro em Ciência é muito pouco eficiente.** Disponível em: < https://www.ufrgs.br/blogdabc/gasto-brasileiro-com-ciencia-e-muito>. Acesso em: 6 de dezembro de 2016.
11. **OpenDOAR.** Disponível em: < http://www.opendoar.org>. Acesso em: 8 de dezembro de 2016.
12. **Bcrypt,** Disponível em: < http://bcrypt.sourceforge.net >. Acesso em: 12 de dezembro de 2016.