Universidade de São Paulo

Instituto de Matemática e Estatística

Trabalho de Formatura Supervisionado

Bacharelado em Ciência da Computação

Refatoração do Projeto Delpo

Adriano Tetsuaki Ogawa Santin

Luiz Fernando Antonelli Galati

Mauricio Luiz Abreu Cardoso

Orientador: Marco Dimas Gubitoso

São Paulo

Fevereiro de 2019

Agradecimentos

1. Adriano

Agradeço à minha esposa Lisandra por todo o amor, apoio e paciência, a meus pais Alvaro (In Memorian) e Toshiko pelos ensinamentos e criação e à minha irmã Viviane por todo o apoio.

2. Mauricio

Agradeço à minha mãe Bete, à minha irmã Denise, à minha filha Sara e a meu pai Jovenil, por todo o apoio e carinho. Graças a vocês tive a oportunidade de passar por esta etapa da vida. Muito obrigado.

3. Luiz

Agradeço ao meu pai, José Luiz, por todo o carinho que me foi dado nesses anos e pelo estímulo ao meu ingresso na faculdade; à minha mãe, Deborah, por todo o amor, carinho, tranquilidade e serenidade a mim transmitidos em cada instante da minha vida; e aos meus amigos mais próximos por todo o apoio que me foi dado quando necessitei.

Resumo

O projeto do Dicionário Etimológico da Língua Portuguesa (DELPo) nasceu da importância de os pesquisadores que se dedicam ao estudo desse idioma contarem com informações a respeito da origem e do significado dos verbetes da língua. O dicionário já está disponível online, mas possui problemas de funcionamento que podem ser atribuídos a falhas no desenvolvimento dos softwares envolvidos no projeto.

Neste trabalho, propusemo-nos a refazer, desde o princípio, grande parte do DELPo (que inclui um complexo banco de dados e vários programas de computador), de modo a produzir uma plataforma que seja estável, relativamente fácil de ser modificada ou aperfeiçoada e cuja interação com os usuários seja intuitiva.

Palavras-chave: dicionário, etimologia, banco de dados, desenvolvimento web.

Abstract

The Dicionário Etimológico da Língua Portuguesa (DELPo) project was born from the need of the reserchers that dedicate themselves to the study of Portuguese language to have information concerning to the origins and the meanings of the words of the language. The dictionary is already available online, but has functional problems that can be attributed to deficiencies on the development of the softwares involved in the project.

In this work, we proposed to redo much of the DELPo system (which includes a complex database and various computer programs) from the begining, in order to create a system that is stable, fairly easy of being modified or improved and whose interaction with users is intuitive.

Keywords: dictionary, etimology, database, web development.

Lista de abreviaturas e siglas

CoC	Convention over Configuration
DELPo	Dicionário Etimológico da Língua Portuguesa
DRY	Don’t Repeat Yourself
Funai	Fundação Nacional do Índio
MVC	Model-View-Controller
RoR	Ruby on Rails
SGBD	Sistema de Gerenciamento de Banco de Dados

Sumário

1 Introdução…….....…….…….....1
1.1 Os estudos etimológicos da língua portuguesa.....1
1.2 A estrutura do NEHiLP e do DELPo.....1
1.3 O funcionamento básico do processo de inserção de palavras no DELPo através do programa Moedor.....4
1.4 O processo de análise etimológica do DELPo.....5
1.4.1 A lexicografia e a definição de lema.....5
1.4.2 Definição de novos termos para o tratamento computacional dos dados.....5

2 Ferramentas utilizadas.....10
2.1 Escolha do *framework*.....10
2.1.1 A linguagem Ruby.....10
2.1.2 O *framework* Ruby On Rails.....11
2.3 Escolha do SGBD.....12
2.4 O Trello.....12

3 Desenvolvimento…….....…….....…….....14
3.1 Refazendo o banco de dados.....14
3.1.2 Dificuldades com o banco legado do DELPo.....15
3.2 Autenticação com devise.....17
3.3 Permissões com CanCanCan.....19
3.4 O programa Moedor.....19
3.4.1 A dificuldade de se trabalhar com código legado.....20
3.4.2 O que o Moedor faz?.....20
3.4.3 Estruturas utilizadas no Moedor.....21
3.4.4 Algoritmo do Moedor.....22

4 Conclusão.....30

Referências bibliográficas.....33

Capítulo 1

Introdução

1.1 Os estudos etimológicos da língua portuguesa

Segundo Viaro et al. (2015, p. 15), “comparado com outras línguas românicas, o português é, de longe, o idioma que tem a informação etimológica mais incompleta e fragmentada. Os dicionários de etimologia têm, com frequência, informação muito vaga e imprecisa sobre as ocorrências mais antigas desses verbetes”.

Tendo isso em vista, foi criado, em 2012, o Núcleo de apoio à pesquisa em Etimologia e História da Língua Portuguesa (NEHiLP). Um de seus projetos é o Dicionário Etimológico da Língua Portuguesa (DELPo), cujo objetivo é satisfazer os requisitos da linguística moderna e sanar deficiências dos atuais léxicos etimológicos. O dicionário, já disponível online, funciona através de um complexo sistema computacional e tem seus dados e todas as modificações em seus verbetes assinadas por pesquisadores, cada um dos quais possui determinado nível de acesso ao projeto mediante *login* e senha pessoais.

Levando em consideração que os *softwares* atualmente responsáveis pelo funcionamento do projeto contêm problemas que são difíceis de serem tratados (como imperfeições na escrita de códigos, ausência de documentação que especifique as atribuições de cada parte do sistema, ferramentas antigas e ineficientes, etc.), propusemo-nos a desenvolver, neste trabalho, um novo DELPo que conte com um sistema computacional estável, consistente e bem documentado. Utilizaremos, para isso, ferramentas como o *framework* Ruby On Rails e o SGBD (Sistema de Gerenciamento de Banco de Dados) PostgreSQL.

1.2 A estrutura do NEHiLP e do DELPo

O NEHiLP tem como objetivo realizar pesquisas acadêmicas sobre Linguística Histórica, Filologia e Etimologia. Para isso, dedica-se ao estudo de documentos das mais variadas épocas, a fim de organizar a informação

linguística do português e gerar dados de qualidade para o estudo daqueles que se interessem pela história da língua portuguesa.

Atualmente, o NEHiLP conta com projetos associados a diferentes universidades: os projetos DBB (Dicionário Dialetal Brasileiro) e DEPARC (Dicionário Etimológico do Português Arcaico), vinculados à UFBA (Universidade Federal da Bahia); o projeto LIBOLO, vinculado à Macau University e à USP (Universidade de São Paulo); o PHPP (Projeto temático História do Português Paulista), vinculado à UNESP (Universidade Estadual Paulista) e à USP; o projeto MAP (Mulheres na América Portuguesa), vinculado à USP; e, por fim, o projeto DELPo (Dicionário Etimológico da Língua Portuguesa), vinculado à USP e objeto de atenção desse trabalho.

O DELPo objetiva ser um dicionário etimológico que esteja disponível online e que ofereça a data da primeira ocorrência das acepções de cada verbete, bem como o contexto dessa acepção. Para isso, o projeto planeja contar um portal online que, fazendo uso de um extenso banco de dados, disponibilize três programas computacionais:

a) o Moedor de Textos: programa que recebe um texto (associado a uma data) e analisa suas palavras, cada uma das quais pode ser marcada em azul, caso ainda não esteja disponível no banco de dados, ou em vermelho, caso já exista no banco de dados e esteja associada a uma data cronologicamente posterior à data do texto analisado;

b) o Metaplasmador: programa que recebe uma palavra em latim vulgar e consegue reproduzir como ela se desenvolveu do latim ao português;

c) o N-GRAM: programa que avalia a quantidade de ocorrências de uma palavra ao longo dos anos.

Tanto o portal online quanto os programas Moedor e N-GRAM já existem, mas contêm imprecisões na escrita dos códigos computacionais e utilizam ferramentas antigas e pouco flexíveis.

O que nos propusemos a fazer nesse trabalho foi:

a) a criação, em linguagem Ruby, de um novo portal do DELPo (ainda que sem grandes elaborações estéticas);

b) a recriação e migração de todo o banco de dados do DELPo (que passou a usar o SGBD PostgreSQL em vez do MySQL);

c) a criação de um mecanismo de autenticação de usuários similar ao que já existina no portal antigo. Esse mecanismo precisou levar em conta a existência de seis níveis de permissões para usuários;

d) a refatoração, em linguagem Ruby, de grande parte do código do programa Moedor, que já existia e estava escrito em linguagem Perl.

O programa Moedor desenvolvido é complexo, pois exigiu o uso de muitas linhas de código e técnicas de programação para ser elaborado. Foi dividido em dois módulos, *moedor* e *analise*.

O módulo *moedor* encarrega-se da parte principal do programa, que, basicamente, consiste em quebrar um texto em contextos e palavras, cada uma das quais será analisada com vistas a encontrar a sua forma básica (por exemplo, a forma básica de *abrindo* é *abrir*) e, então, comparada com o banco de dados para estabelecer a data mais longínqua que pode ser associada a ela levando-se em consideração as informações disponíveis até então. Os dados obtidos por esse módulo do programa são organizados e armazenados em uma tabela do banco chamada *resultados*.

Já o módulo *analise* é uma continuação do módulo *moedor*. Apesar de o programa Moedor ter sido criado para automatizar a análise etimológica de palavras da língua portuguesa, ainda é imprescindível que haja um olhar humano para vistoriar tudo o que passa pelo sistema. Por isso, o módulo *analise* consulta os dados na tabela *resultados* (criada pelo módulo *moedor*) e apresenta-os ao pesquisador utilizando a forma definida acima, na descrição do programa Moedor. Assim, cada palavra do texto inserido pode ser mostrada com uma marcação em azul, caso ainda não esteja disponível no banco de dados; ou em vermelho, caso já exista no banco de dados e esteja associada a uma data cronologicamente posterior à data do texto analisado.

1.3 O funcionamento básico do processo de inserção de palavras no DELPo através do programa Moedor

Para explicar como ocorre o processo de inserção de palavras no dicionário através do programa Moedor, vamos utilizar a seguinte definição:

Dizemos que uma data a no formato dd/mm/aaaa é maior do que outra data b no mesmo formato se o dia especificado por a tiver ocorrido depois do especificado por b.

Para adicionar palavras no dicionário, é necessário que um pesquisador submeta uma obra, juntamente com sua data, ao Moedor do DELPo. Cada palavra presente no texto inserido é analisada e, então, uma das seguintes ações é executada:

a) se a palavra não estiver no banco de dados, o moedor marca-a como "inexistente", associa a ela a data da obra submetida e apresenta-a ao pesquisador, que decidirá se o conjunto formado pelo vocábulo e pela data a ele associada deve ser incluído no banco;
b) se a palavra já existir no banco de dados e estiver vinculada a uma data maior do que a da obra submetida, ela (a palavra) é marcada como "retrodatou" e é associada à data do texto inserido. Também nesse caso, o vocábulo é apresentado ao pesquisador, que decidirá se deve autorizar a atualização da data vinculada a ele no banco.

Dessa maneira, ao final da inserção de uma ou mais obras, a cada palavra do banco de dados estará associada uma data que representará o dia da primeira ocorrência do vocábulo conhecida até então. Quanto mais obras forem submetidas ao moedor, maiores serão as chances de as datas associadas às palavras retratarem o dia exato em que cada uma delas ocorreu pela primeira vez em um texto da língua portuguesa.

1.4 O processo de análise etimológica do DELPo

1.4.1 A lexicografia e a definição de lema

Segundo Salviano (2014), “a lexicografia é a ciência responsável pelo desenvolvimento de métodos e técnicas de produção das obras dicionarísticas na sua variedade de formas (monolíngues, bilíngues, semibilíngues, escolares, gerais, infantis, etc)”. Assim, o trabalho de elaboração dos mais variados tipos de dicionários pertence ao campo da lexicografia.

Dentro dessa ciência, *lema* é uma unidade básica dotada de significado. Assim, num determinado dicionário, todo signo que seja objeto de definição ou explicação é um lema. Todo vocábulo descrito em um dicionário de palavras da língua portuguesa, como, por exemplo, o Houaiss ou o Aurélio, é, portanto, um lema. Já em um hipotético dicionário de siglas relacionadas ao mundo da tecnologia e da computação, o signo “SGBD”, por exemplo, seria um lema.

Dentro do campo da lexicografia, explica Viaro (2017, p. 146) que “duas palavras homógrafas podem ser, dependendo do caso, dois lemas ou um só, dependendo se são, respectivamente, fruto de homonímia (etimologias distintas) ou de polissemia (têm uma origem comum)”.

1.4.2 Definição de novos termos para o tratamento computacional dos dados

Para a análise etimológica da língua, o DELPo utiliza a seguinte hierarquia de dados:

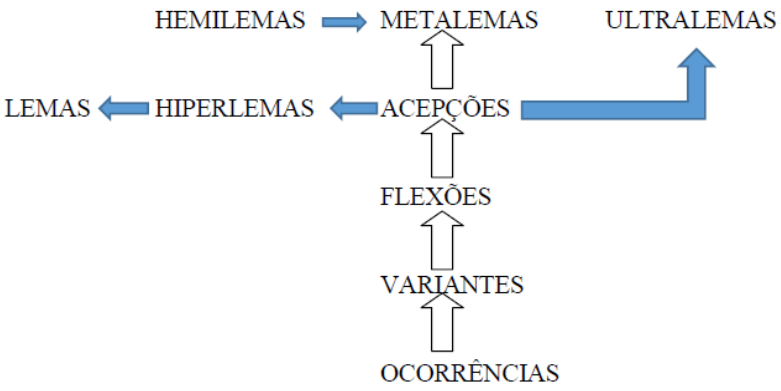

Figura 1.1: Hierarquia de dados do DELPo

Metalema*, hemilema, hiperlema* e *ultralema* são termos criados especificamente para facilitar o tratamento computacional dos dados que serão recebidos pelos sistema DELPo.

Metalema é simplesmente uma sequência de letras única e dotada de significado, não importa a quantas acepções corresponda. Não existem metalemas homônimos. Isso difere do lema tradicional lexicográfico, já que, nesse caso, como já explicado, duas palavras homógrafas podem pertencer a dois lemas ou somente a um.

Assim, num dicionário da língua portuguesa, a palavra *manga*, por exemplo, pode estar associada a dois lemas: *manga*[1] (fruta) e manga[2] (parte do vestuário). Entretanto, não é possível, para o programa Moedor do DELPo, separar automaticamente essas sequências de caracteres em dois lemas distintos, pois, para isso, seria necessário que os significados associados às palavras fossem identificados automaticamente pelo *software*. Desse modo, dentro do programa, tanto *manga* (fruta) quanto *manga* (parte do vestuário) são classificadas como um único metalema, e a atribuição de sentido a cada um dos vocábulos *manga* que aparecem no texto deve ser feita posteriormente por um pesquisador. Isso, é claro, vale para todas as palavras que aparecem na obra analisada.

Para encontrarmos o metalema de uma palavra, adotamos a mesma metodologia que a lexicografia tradicional adota para o lema e, assim, o metalema:

- de um verbo é sua forma no infinitivo;
- de um substantivo é sua forma no singular;
- de um adjetivo é sua forma no masculino singular;
- é sempre adotado em sua ortografia atual.

Portanto, o verbo flexionado *cantava*, por exemplo, deve estar associado ao metalema *cantar*; o substantivo *mesas*, ao metalema *mesa*; o adjetivo *bonitas*, ao metalema *bonito*; e a palavra *heróico*, ao metalema *heroico.*

Hemilema é uma palavra que, para ser decifrada, exige a interpretação da acepção de um metalema. É uma denominação usada para tratar abreviaturas em geral e palavras que podem estar truncadas devido ao mau estado de conservação do manuscrito.

Consideremos, por exemplo, as abreviaturas *mtos* e *mta*: ambas são interpretadas como uma das acepções do metalema *muito* e, portanto, não remetem a *muitos* e *muita*, suas respectivas flexões.

É importante notar que uma sigla, apesar de ser um tipo especial de abreviatura, será tratada como um metalema, e não como um hemilema, pois há siglas que se assemelham a nomes comuns (*Ibope*, por exemplo) e só um pesquisador conseguirá classificá-las corretamente.

O *hiperlema* foi criado para que, ao analisarmos os dados, pudéssemos recuperar as condições de homonímia e polissemia, ausentes na constituição do metalema. Para entendermos melhor como o conceito de hiperlema é utilizado, precisamos definir alguns termos referentes às acepções.

Uma acepção é subordinada se remete a uma outra acepção, chamada subordinante. Se uma acepção subordinante não for ao mesmo tempo subordinada, ela é chamada de acepção principal. Toda acepção principal gera um hiperlema. Um metalema, por sua vez, pode ter duas ou mais acepções principais e, nesse caso, gerará hiperlemas homônimos, o que nos possibilita fazer a distinção entre homonímia e polissemia de um único metalema.

Ultralema é um conceito criado para tratar radicais, raízes e elementos de formação de uma palavra (como prefixos e sufixos). A palavra *caligrafia*, por exemplo, possui dois radicais (*cali* e *grafia*) e, portanto, dois ultralemas – um para cada radical. Já a palavra *bigamia* tem um prefixo, *bi-*, e um radical, *gamia*, e, portanto, também possui dois ultralemas – um para o prefixo e outro para o radical.

Um ultralema adota as seguintes formas:

- -x, quando é um sufixo final;
- -x-, quando é um sufixo não final (requer uma vogal temática, por exemplo);
- x-, quando é um prefixo;
- x, sem os hifens, nos demais casos (quando é raiz, por exemplo).

Capítulo 2

Ferramentas utilizadas

2.1 Escolha do *framework*

Um *framework* é um conjunto de bibliotecas e ferramentas capazes de facilitar e agilizar a criação e o desenvolvimento de um sistema computacional. É algo necessário para que a construção de uma aplicação *web* - como a desenvolvida nesse trabalho - ocorra com rapidez.

Tínhamos como opções principais de *frameworks* o JSF (Java Server Faces), o NodeJs, o Django e o Ruby On Rails. Escolhemos este último por conta das vantagens oferecidas pela linguagem Ruby e pelo *framework* Ruby on Rails, as quais descrevemos a seguir.

2.1.1 A linguagem Ruby

A linguagem *back-end* do Rails é o Ruby, que possui uma grande capacidade de tratar expressões regulares, o que é importantíssimo para a análise de palavras. Ruby possui forte influência da linguagem Perl (na qual foi escrito o código original do programa moedor). Isso pode ser visto, por exemplo, através da observação do fato de que Ruby declara seus objetos como *hashtables*, de modo análogo ao que ocorre em Perl, o que permite realizar mudanças na estrutura de objetos sempre que necessário. Esse recurso foi muito utilizado no código legado do moedor.

Além disso, Ruby é uma linguagem simples de ser entendida (característica importante em um projeto no qual novos desenvolvedores podem entrar a qualquer momento), orientada a objetos (possibilitando que trechos de código possam ser reutilizados várias vezes), implicitamente tipada (ou seja, os tipos de variáveis não precisam ser declarados) e dinamicamente tipada (pois permite que o tipo de uma variável possa ser alterado durante a execução do programa). Tudo isso torna o desenvolvimento em Ruby fácil e rápido.

Por fim, cabe assinalar ainda que as bibliotecas em Ruby são conhecidas como *gems*, e há um grande acervo em seu repositório, tornando fáceis tarefas como autenticação, encriptação, implementação de páginas *web* com javascript, etc.

2.1.2 O *framework* Ruby On Rails

Uma das características do Rails que o torna atrativo a projetos que são de longo prazo e que pretendem contar com diversos desenvolvedores é a possibilidade de realizar mudanças de maneira incremental. Isso significa que um membro do projeto não necessariamente precisa entender a fundo tudo o que já está feito para que possa implementar novas funcionalidades e criar novas estruturas.

Dois pilares desse *framework* são os conceitos *Don't Repeat Yourself (DRY)* e *Convention Over Configuration (COC).*

O *DRY* é um princípio de desenvolvimento de software que objetiva diminuir a reescrita de trechos de código que realizam tarefas parecidas. A ideia é reaproveitar o máximo possível aquilo que já foi elaborado no sistema. O Rails possui vários recursos úteis para ajudar os desenvolvedores a colocar esse princípio em prática.

Já o *COC* é um paradigma de desenvolvimento de sistemas que define regras e padrões (de nomes e de localização de arquivos, por exemplo) que auxiliam desenvolvedores a diminuir o tempo utilizado para tomar decisões de caráter organizacional. O Rails, no entanto, permite que esses padrões sejam quebrados caso necessário, tornando-o também flexível e adaptável.

Além disso, esse *framework* utiliza um padrão de arquitetura conhecido como *MVC (Model-View-Controller),* que organiza as funcionalidades do sistema em três camadas bem delimitadas: a *View*, responsável pela apresentação de requisições e resultados na página web; a *Controller*, responsável por gerir, controlar e direcionar todas as ações executas no sistema; e a *Model*, responsável pelo armazenamento dos dados necessários para o funcionamento do sistema.

2.3 Escolha do SGBD

Havia duas opções principais entre os dois maiores bancos *open-source* relacionais disponíveis na atualidade: PostgreSQL e MySQL. O banco do projeto antigo utilizava MySQL, entretanto optamos pela mudança para PostgreSQL, pois este possui uma gama forte de operações, é um banco bem consolidado e, principalmente, suporta cargas maiores de dados, o que é necessário em um projeto de longo prazo que receberá muitas obras.

2.4 O Trello

O Trello é uma ferramenta para auxiliar no gerenciamento, organização e coordenação de projetos. Facilita o compartilhamento de informações, a comunicação e a divisão clara de tarefas entre os desenvolvedores.

Ao funcionar como uma espécie de espaço de armazenamento de informações úteis para a refatoração do DELPo, o Trello permitiu que nós, programadores, não perdêssemos tempo consultando um ao outro para saber o que já havia sido feito e quais tarefas ainda não haviam sido executadas.

Capítulo 3

Desenvolvimento

3.1 Refazendo o banco de dados

Uma das partes mais importantes do sistema DELPo é o moedor de textos, que analisa, uma a uma, as palavras e seus contextos, e associa a elas todas as estruturas relacionadas (acepção, lema, metalema, hemilema, hiperlema, ultralema, flexão), auxiliando, assim, na datação e análise etimológica da língua portuguesa.

Toda essa informação é salva em um banco de dados que possui tabelas para cada uma das estruturas linguísticas ou computacionais descritas na seção 1.4.2.

As tabelas de banco de dados que já existiam no antigo sistema DELPo podem ser vistas na figura abaixo:

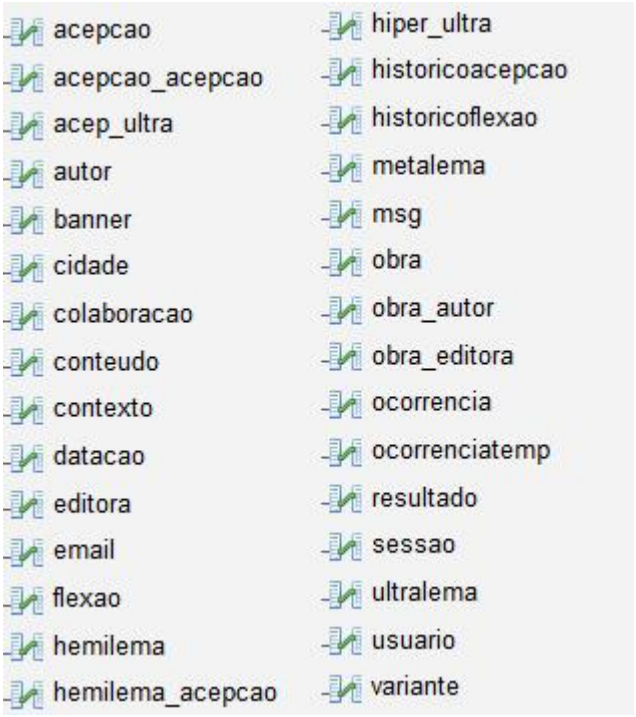

Figura 3.1: Tabelas do antigo sistema DELPo

Parte crucial no trabalho de refatoração do sistema DELPo foi a recriação e a adaptação desse banco de dados, ferramenta de suma importância para o bom funcionamento do projeto e que, por isso, necessita ser mantida bem estruturada, limpa (sem informações divergentes, que apontam para dados inexistentes), e fácil de entender. O *framework* Ruby On Rails auxilia muito esse processo, na medida em que permite criar e modificar facilmente estruturas de banco de dados.

As novas tabelas foram criadas através desse *framework* usando um sistema chamado *Active Record*, que é uma ferramenta do RoR que gerencia dados e lógica. Com ela, é possível conectar objetos complexos do sistema a tabelas em um banco de dados relacional, e as propriedades e relações dos objetos são facilmente armazenadas e acessadas, sem o uso de extensos comandos SQL para acesso ao banco de dados, que, assim, se torna mais simples de ser entendido, gerenciado e mantido a longo prazo.

3.1.1 Dificuldades com o banco legado do DELPo

No RoR, ao iniciarmos um novo projeto e especificarmos que usaremos o SGBD PostgreSQL, os bancos de dados (três, no total: um para testes, um para desenvolvimento e um para produção) são criados e já ligados ao Rails. Para a criação das novas tabelas, tivemos que analisar as existentes do banco anterior e construir uma por uma. No Rails, é possível criar cada tabela com um simples comando em que definimos o nome da tabela, suas colunas associadas a um determinado tipo de dados e as chaves estrangeiras.

Tivemos problemas no momento de definir os relacionamentos entre as tabelas (pois eles estavam mal especificados no banco de dados anterior) e qual a ordem em que elas deveriam ser criadas, já que, para a criação de chaves estrangeiras, a tabela para a qual a chave aponta já deve existir.

Este foi o comando genérico usado para a criação das tabelas:

```
rails generate model nome_da_tabela \

nome_da_coluna_1:tipo_da_coluna_1 \
```

```
nome_da_coluna_2:tipo_da_coluna_2 \

chave_estrangeira:references
```

Com esse comando, é relativamente simples criar uma tabela. No caso da existência de uma chave estrangeira apontando para uma segunda tabela (chamada, por exemplo, “externa”), é só usar a expressão “references” como o tipo da coluna:

```
externa:references
```

Isso criará uma coluna chamada “externa_id” com valores que se relacionam com o campo *id* da tabela *externa*.

Após a criação das tabelas no novo banco de dados, foi necessário fazer a migração dos dados do banco antigo para o novo, o que parecia ser algo simples, já que é possível exportar os dados de um banco em um arquivo SQL já preparado para a inseri-los em um novo banco (desde que este tenha a mesma estrutura do anterior). Foi aí que começaram as dificuldades: o banco novo, apesar de seguir a estrutura do antigo, possuia vários detalhes distintos, o que tornou a migração de dados algo não tão trivial. Essas diferenças se deram por causa das decisões de projeto e tecnologias usadas no novo sistema, que diferiam do anterior.

A primeira diferença foi que o Rails utilizou, como é padrão no *framework*, o plural para dar nome a tabelas do banco, enquanto os nomes das originais estavam no singular. O Rails também criou automaticamente duas colunas a mais em cada tabela: a *created_at* e a *updated_at,* que armazenam os momentos de criação e de alteração de cada entrada no banco. Tais colunas não existiam no banco original.

Outros problemas surgiram devido à mudança do SGBD (de MySQL para PostgreSQL). Para colunas do tipo *boolean,* o MySQL aceitava os valores 0 e 1 como *false* e *true,* respectivamente, enquanto o PostgreSQL não os aceitava. Do mesmo modo, MySQL aceita entradas de datas inválidas, como “0000-00-00”, enquanto o PostgreSQL, não.

Portanto, embora tenha sido possível extrair um arquivo de extensão *.sql* que fizesse a migração dos dados automaticamente, foi necessário editá-lo várias vezes para corrigir cada uma das incoerências descritas acima. Depois de corrigidas, o Rails conseguiu rodar esse arquivo e fazer todas as inserções de modo rápido, com um simples comando:

```
bundle exec rails db < arquivo.sql
```

Ao rodar esse comando com o arquivo, também foram encontradas algumas inconsistências que estavam presentes no banco antigo, como chaves estrangeiras que apontavam para entradas inexistentes. Tais erros foram corrigidos com a eliminação das entradas inconsistentes, e o processo de migração foi finalizado.

3.2 Autenticação com devise

Grande parte dos serviços *web* existentes exigem autenticação a fim de proteger suas informações ou mesmo de segregar escopo de acesso para cada usuário. No caso do projeto DELPo, todo usuário tem associado um inteiro de 0 a 5, que representa seu nível e suas permissões, sendo 0 o menor conjunto de permissões possíveis, e 5 o maior conjunto de permissões possíveis. A inclusão de outros usuários no sistema é feita por intermédio de um usuário de nível 5, pois somente com este nível é possível adicionar usuários.

A *gem* de autenticação e sessão mais famosa e bem consolidada no mundo Rails é a *devise*. Essa *gem* já vem com um arcabouço de funcionalidades prontas, como interfaces para o próprio usuário se inscrever no sistema e envio automático de e-mails após a inscrição. Tudo isso é embarcado e não possui grande dificuldade de adaptação ao sistema por parte do desenvolvedor. Entretanto, a modificação dessas funcionalidades para que um usuário adicionasse o outro provou-se difícil.

Após muitos conflitos devido às definições do Rails, foi decidido que as rotas do *devise* de sessão, registro, etc. viriam primeiro, seguidas pelas rotas comuns dos usuários, tendo sido criado o controlador *registrations* apenas para adicionar a verificação de permissão do CanCanCan, *gem* que gere o controle de acesso e que será citada adiante. O motivo dessa ordem de rotas é o *devise* não permitir que um usuário com sessão adicione outro usuário. Logo, as rotas padrão de controle de usuários contornaram o problema.

```ruby
Rails.application.routes.draw do

  devise_for :usuarios, controllers: {registrations: 'usuarios/registrations'}
  resources :usuarios
  root 'pages#index'
  resources :obras
  resources :autores
  resources :editoras
  resources :cidades
  resources :moedor, only: [:new]
  post 'moedor/moer'
  # For details on the DSL available within this file, see http://guides.rubyonrails.org/routing.html
end
```

Figura 3.2: Rotas do *devise* e dos usuários

A tabela do usuário precisou ser gerada pelo próprio *devise*, tendo sua estrutura posteriormente alterada com a adição das colunas nível, e-mail, etc. Após isso, foi necessário filtrar todas as chamadas *http* do site para adicionar verificação de sessão, ou seja, para o usuário ter a oportunidade de *logar* se não estiver *logado*.

Foi necessário, ainda, que todos os controladores herdassem o parâmetro *authenticate_usuario!* do *devise*, para permitir que os usuários pudessem ser redirecionados à tela de login. Também foi utilizado o *sanitizer*, para possibilitar ao usuário parâmetros diferentes de e-mail e senha (ambos default do *devise*) - no nosso caso: *level*, filologia, etc.

```ruby
class ApplicationController < ActionController::Base
  before_action :authenticate_usuario!, except: [:index]
  before_action :configure_permitted_parameters, if: :devise_controller?


  protected

  def current_ability
    @current_ability ||= Ability.new(current_usuario)
  end

  def configure_permitted_parameters
    devise_parameter_sanitizer.permit(:sign_up, keys: [:username, :level, :instituicao, :lattes, :abreviatura, :filologia, :edicao, :programador, :produtivo, :nome])
  end

  rescue_from CanCan::AccessDenied do |exception|
    redirect_to '/', :alert => 'Você não tem permissão'
  end
end
```

Figura 3.3: autenticação do *devise* e *sanitizers*

3.3 Permissões com CanCanCan

A *gem* CanCanCan é responsável pelo controle de permissões. Ela permite ocultar informações ou links da *view*, e também proteger o *back-end* caso um usuário mal-intencionado tente acessar um recurso sem permissão, utilizando um *request* bem formado. Tal *request* é barrado no *front* e também no *back-end* pelo CanCanCan. A política de nível é implementada no modelo *ability.rb*. A permissão de todo o *website* é definida nesse arquivo. No momento em que escrevemos, é necessário ter nível 5 para visualizar, incluir ou alterar linhas do banco de dados, sendo todos os outros níveis barrados para as mesmas funções. Segue imagem do *ability.rb* atual:

```
class Ability
  include CanCan::Ability

  def initialize(usuario)
      # can :manage, Post, user_id: user.id
      if usuario.level >= '5'  # additional permissions for administrators
        can :manage, :all
      else
        cannot :manage, :all
      end
    else
        cannot :manage, :all
    end
  end
end
```

Figura 3.4: gestão de permissão do *ability.rb*

Foi inserido, em cada controlador que se relaciona a tabelas do banco, o parâmetro *load_and_authorize_resource*, para que seja verificada a permissão do usuário.

3.4 O programa Moedor

O sistema DELPo tem como objetivo auxiliar na pesquisa etimológica da língua portuguesa. Esse tipo de pesquisa depende grandemente da coleta exaustiva e da datação precisa de dados em forma de textos. Portanto, no centro do sistema DELPo encontra-se um algoritmo computacional capaz de ler os textos, coletando e classificando todos os dados que serão utilizados na pesquisa. Esse algoritmo é chamado de Moedor.

O código original do Moedor encontrava-se inteiramente escrito na linguagem Perl, porém, ao fazer uma refatoração do DELPo no *framework* RoR, foi feita, naturalmente, uma refatoração do Moedor, na linguagem Ruby, integrando-o perfeitamente com o resto do sistema.

3.4.1 A dificuldade de se trabalhar com código legado

Alterar código legado é sempre uma dificuldade, seja pela tecnologia que mudou, como uma biblioteca ultrapassada, ou pelo fato de a lógica de negócio ter sido implementada por um programador que não faz mais parte da equipe. No DELPo, além desses problemas comuns, nos foi imposta a necessidade de lidar com linguagens de programação diferentes da linguagem anterior do projeto, que tinha a interface *web* em PHP, o Moedor em Perl, e as *querys* do banco em linguagem SQL nativas, sendo essas 3 linguagens inexistentes no projeto atual.

Foi necessário, consequentemente, três curvas de aprendizado, uma relativa ao aprendizado das tecnologias anteriores, outra relativa ao aprendizado da tecnologia atual e outra relativa à absorção da lógica de negócio. Percebeu-se também que a ausência de comentários no código original aumentou acentuadamente a dificuldade de transposição do código legado.

3.4.2 O que o Moedor faz?

Ao acessar o sistema do DELPo, um pesquisador pode submeter uma obra para ser analisada, assim como sua data de origem. O Moedor, então, lê cada uma das palavras do texto, coletando os contextos delas e quebrando-as em suas formas primitivas, de modo a tentar entender quais são exatamente os termos (e acepções dos termos) utilizados no texto.

Juntamente com essa quebra, o Moedor conta com o banco de dados do DELPo para comparar com as palavras que analisa e, com esses dados, é

possível determinar, de forma mais precisa, a cada nova obra inserida no banco, a datação e análise etimológica da palavra.

Conforme o texto passa pela moagem (processo de passar pelo moedor), palavras que são consideradas importantes para a pesquisa etimológica são selecionadas e, ao final da moagem, essas palavras são listadas para o pesquisador de forma organizada, juntamente com o contexto em que aparecem e suas datações, separadas por cores, para diferenciar se foi encontrada uma data mais antiga para cada uma das palavras ou não.

3.4.3 Estruturas utilizadas no Moedor

Uma vantagem que a linguagem Ruby tem sobre a linguagem Perl é o fato de ser orientada a objetos. Por isso, pudemos construir o Moedor como uma classe e, assim, para cada texto inserido no sistema, uma instância do Moedor é criada, e em sua estrutura todas as informações da moagem são armazenadas (como atributos do objeto) antes de passarem para o banco de dados.

Primeiramente, todos os textos submetidos ao Moedor devem estar em arquivos no formato *txt* codificados em UTF-8 (o motivo dessa codificação será explicado posteriormente) e com até 2 megabytes de tamanho. As datas dos textos podem estar em vários formatos, como só o ano (1970), a data completa (01/08/1970) ou até mesmo somente o século (XIX). O moedor faz uma análise da data e a padroniza antes da moagem.

O texto do arquivo é convertido em uma *string* única, sem quebras de linha, mas não é processado desse modo: é dividido em contextos, cada um dos quais é processado palavra por palavra. Originalmente, a ideia era separar o texto em linhas em vez de contextos, porém isso poderia gerar problemas dependendo do formato do texto original e de como as linhas eram separadas. Decidiu-se então pela divisão em contextos, definidos como o texto entre duas pontuações (ponto final, ponto de interrogação ou ponto de exclamação).

Após a divisão do texto, os contextos são armazenados em dois *arrays*: um chamado *contexto* e outro chamado *contexto_final*. O primeiro *array*

mantém os contextos nas suas formas originais, e o segundo é usado na análise e modificado durante o processo.

O processo de divisão é repetido para os contextos: cada um deles é quebrado, para análise, em palavras, que, depois desse processo de moagem, são armazenadas não em um array, mas em um *hash* que as leva a listas com os contextos em que aparecem.

Essas são as estruturas básicas, porém outras são utilizadas. Existem listas de abreviaturas conhecidas: *homônimos*, *antropônimos*, *topônimos* e *algarismos romanos*, todas com o propósito de auxiliar na comparação das palavras e na determinação exata de seu significado e de sua etimologia com o mínimo de erros.

Outras duas *hashes* foram utilizadas: uma chamada *color* e outra chamada *taq*.

Color relaciona uma palavra a uma classe. Cada palavra é analisada com referência às suas propriedades e também às suas datações atuais (comparando a data da obra inserida com as de outras obras em que a palavra foi encontrada). Desse modo, as palavras são divididas em classes e separadas por cores (para facilitar a análise posterior pelos pesquisadores).

Taq, cujo nome vem do termo em latim "*terminus ante quem*", quer dizer a data antes da qual um documento deve ter sido escrito. Essa *hash* é utilizada para associar uma palavra à sua data mais antiga já encontrada. O valor é encontrado comparando as datas da obra submetida com todas as outras datações dessa palavra em outras obras no sistema.

3.4.4 Algoritmo do Moedor

Ao receber o texto, a primeira coisa que o Moedor faz é tratar as marcações feitas pelos pesquisadores. Os textos originais são transcritos digitalmente pelos pesquisadores, que utilizam marcações pré-definidas para destacar partes específicas do texto.

As marcações principais são as seguintes:

- [[segmento]] - um segmento de texto entre colchetes duplos indica que ele deve ser ignorado, seja por ser um comentário do pesquisador ou por qualquer outro motivo. Dentro de um contexto, esse segmento é simplesmente removido antes da análise;
- {segmento} - um segmento entre chaves indica que ele deve ser tratado literalmente. Esse trecho é adicionado ao bancos de dados na forma em que se encontra, sem alterações;
- #segmento# - um segmento entre cerquilhas indica que ele estava riscado no texto original. Nesse caso, as palavras do segmento são analisadas normalmente, mas, ao final, para manter o contexto em sua forma original, o texto é exibido de forma tachada: ~~exemplo~~.

Outra marcação que é utilizada é a do “ponto fantasma”. Ele é representado no texto por um ponto final entre chaves: [.]. Como dito anteriormente, contextos são definidos entre pontuações. Porém, em alguns casos, contextos podem ficar muito longos, ou nem existir uma divisão em contextos (como no caso de poemas, que não possuem pontuação). Nesses casos, os pesquisadores podem adicionar esses pontos fantasmas em lugares específicos para que o moedor possa efetuar a divisão em contextos de forma mais eficaz.

As últimas duas marcações são o “espaço adicionado” e o “espaço removido”. O “espaço adicionado” refere-se a duas palavras ligadas por um *underline* (*estava_almoçando*, por exemplo), e é usado pelo pesquisador para representar um espaço que não existia no texto original, mas que o Moedor precisa ver de modo a separar as duas palavras. Já o “espaço removido” refere-se a dois segmentos de texto com um sinal de igual entre eles (condici=onador), e é usado pelo pesquisador para representar um espaço que existia no texto original, mas que não deveria existir, já que as partes representam uma única palavra. Esse segundo caso costuma acontecer no fim de uma linha em textos manuscritos antigos.

Todas essas marcações, porém, são coisas adicionadas pelos pesquisadores e não se encontram no texto original. É para manter a

integridade do texto que são criadas duas cópias de cada contexto, e é necessário que o texto esteja encodado em UTF-8.

Ambas as cópias do contexto são analisadas em busca dessas marcações. Uma das cópias trata as marcações de modo a retornar ao formato original do texto: os espaços voltam ao lugar em que estavam, palavras são tachadas, etc.

Na outra cópia, as marcações são tratadas de modo que o contexto fique mais claro para o Moedor tratar, separando as palavras da forma correta e sumindo com segmentos desnecessários. É preciso, porém, manter as marcações para saber como tratar cada parte diferente. Por isso, as marcações são substituídas por caracteres especiais em UTF-8. Tais caracteres não ocupam espaço (se estiverem em qualquer lugar do texto, ao imprimir o texto, não vão aparecer visualmente), mas são lidos e interpretados pelo Moedor, identificando todas essas partes.

Após o tratamento das marcações do texto, a análise das palavras é iniciada. Para cada contexto, as palavras são tomadas uma a uma e passam por vários testes nos quais suas características são analisadas de modo a reduzi-las às suas formas mais puras (formas de dicionário).

Nesses testes, são utilizadas expressões regulares para identificar certos padrões nas palavras. Expressões regulares são extremamente eficazes para se analisar um texto. Em Ruby, é possível identificar padrões no texto, armazenar partes específicas desses padrões em variáveis, e substituir padrões de modo fácil. Expressões regulares são encapsuladas entre barras em Ruby:

```
match = palavra.match(/(\w+)-(se)-([mt]e|lhes?|[nv]os)$/)

s1, s2, s3 = match.captures
```

O exemplo acima é uma captura de expressão regular em uma *string* armazenada em uma variável “palavra”. Nesse exemplo, tenta-se pegar uma mesóclise do tipo -se-lhe ou -se-vos (por exemplo, “Mostravam-se-lhe os pés”, Machado de Assis). Depois de encontrar o padrão na palavra, o método

captures consegue separar em diferentes variáveis as partes delimitadas em parênteses na expressão, isolando o verbo.

Várias expressões são utilizadas para cada caso diferente:

```
palavra.match(/(\w+)-(lhe)-(l[oa]s?)$/)
palavra.match(/(\w+)l-([oa]s?)$/)
palavra.match(/(\w*?mo)-(l[oa]s?|nos?)$/)
palavra.match(/(\w*?mo)-no-(l[oa]s?)$/)
palavra.match(/(\w+?)-([nv][oa])-(l[oa]s?)$/)
palavra.match(/(\w*?fi)-(l.{1,2})$/)
palavra.match(/(\w*?f)ê-(l.{1,2})$/)
palavra.match(/(\w*?f)á-(l.{1,2})$/)
palavra.match(/(\w*?tr)([áa])-(l[oa]s?)$/)
palavra.match(/(\w*?)ê-(l[oa]s?)$/)
palavra.match(/(\w*?e)-(l[oa]s?)$/)
palavra.match(/(\w*?fi)-(l.{1,2})$/)
palavra.match(/(\w*?f)ê-(l.{1,2})$/)
palavra.match(/(\w*?ai)-(l[oa]s?)$/)
palavra.match(/(\w*?)ô-(l[oa]s?)$/)
palavra.match(/-(lhes?|[mt]e|[mtnv]?[oa]s?)$/)
palavra.match(/-(lh[oa]s?)$/)
palavra.match(/-se$/)
```

As palavras primeiramente são testadas para o caso de serem mesóclises ou ênclises (com as expressões acima, porém não se encontram aqui todas as expressões usadas), pois, nesses casos, podem ser consideradas como mais de uma palavra. Para essas, o pronome é separado da palavra e adicionado ao fim do contexto para análise posterior, enquanto a forma verbal continua a sua análise de forma individual, identificando o verbo na forma infinitiva.

Em seguida são testadas para os casos de palavras compostas, que devem ser analisadas como uma única palavra.

Após esses casos específicos, palavras são tratadas de modo a remover superlativos, aumentativos, diminutivos e plurais (como nos exemplos abaixo) e, nesse processo, já é definida a classe da palavra.

```
palavra.match(/(.+)ã[oe]zinh([ao])s$/)
palavra.match(/(.+)õezinh([ao])s$/)
palavra.match(/(.+[qg]uizinh[ao])s$/)
palavra.match(/(.+[aeou])izinh([ao])s$/)
palavra.match(/(.+)re*zinh([ao])s$/)
palavra.match(/(.+[sz])ezinh([ao])s$/)
palavra.match(/(.+)õezonas$/)
palavra.match(/(.+)ã[oe]zãos$/)
palavra.match(/(.+[aeou])izões$/)
palavra.match(/(.+)rzãos$/)
```

As classes definidas no sistema são:

- Homonímia
- Nome Próprio
- Flexão
- Composto
- Advérbio de modo
- Diminutivo
- Aumentativo
- Superlativo
- Abreviatura

Então, as palavras são comparadas com bancos de dados e listas de palavras existentes no sistema para identificar seu tipo e sua flexão. Depois disso, é realizado o processo de datação. Cada uma das palavras, já na forma sua mais básica, pode ser classificada como *inexistente* ou *retrodatou*.

Se uma palavra é classificada como “inexistente”, significa que ainda não está no banco de dados; se uma palavra é classificada como “retrodatou”, significa que ela já existe no banco de dados associada a uma data maior do que a data da obra sendo analisada (o conceito de “maior” segue a definição dada na seção 1.3). Em ambos os casos, o vocábulo é associado à data da obra em análise.

Então, todos esses dados (as palavras nas suas formas básicas, os contextos, as datas e as classificações encontradas) são salvos em uma tabela chamada *resultados*, e o módulo *analise* (a segunda parte do programa Moedor) é chamado.

Por fim, o módulo *analise*, de posse desses dados, cria uma tabela para ser mostrada ao usuário, que é formatada dando cores a cada palavra, indicando sua classificação e mostrando de modo simples os resultados da moagem. O pesquisador que está utilizando o Moedor do Delpo pode analisar as palavras e os resultados e decidir se as palavras realmente foram analisadas, classificadas e datadas corretamente.

Segue um exemplo de como os resultados são visualizados:

além	Retrodatou	de palha não me deixava a menor esperança.	1894
estar	Retrodatou	Além de estar escura a noite, um maldito véo cahido de um chapéozinho de palha não me deixava a menor esperança.	1894
escura	Inexistente	Além de estar escura a noite, um maldito véo cahido de um chapéozinho de palha não me deixava a menor esperança.	1894
maldito	Retrodatou	Além de estar escura a noite, um maldito véo cahido de um chapéozinho de palha não me deixava a menor esperança.	1894
palha	Retrodatou	Além de estar escura a noite, um maldito véo cahido de um chapéozinho de palha não me deixava a menor esperança.	1894
menor	Retrodatou	Além de estar escura a noite, um maldito véo cahido de um chapéozinho de palha não me deixava a menor esperança.	1894
melhor	Retrodatou	Resignei-me, e assentei que o melhor era cuidar de outra cousa.	1894
cuidar	Retrodatou	Resignei-me, e assentei que o melhor era cuidar de outra cousa.	1894
outra	Retrodatou	Resignei-me, e assentei que o melhor era cuidar de outra cousa.	1894
pensamento	Retrodatou	Já o meu pensamento tinha-se lançado á galope pelo mundo da fantasia, quando de repente foi obrigado á voltar por uma circumstancia bem simples.	1894
lançado	Retrodatou	Já o meu pensamento tinha-se lançado á galope pelo mundo da fantasia, quando de repente foi obrigado á voltar por uma circumstancia bem simples.	1894

Figura 3.5: o resultado de um processo de moagem

A captura de tela acima representa o resultado apresentado por nosso sistema quando o rodamos com a obra “Cinco minutos” de José Martiniano de

Alencar, de 1894. A começar pela esquerda, as colunas representam: a palavra, sua classificação, o contexto em que aparece no texto e o ano (ou data). As cores representam as classificações, de modo a facilitar a identificação visualmente.

Todo o módulo *moedor* encontra-se funcionando em integração com o banco de dados que migramos para PostgreSQL e que é acessado pelo *Active Record* do Rails. Já o módulo *analise* é capaz de consultar o banco e criar a tabela para visualização.

Capítulo 4

Conclusão

A realização de um trabalho extenso e complexo como este só foi possível graças à existência de uma equipe de coordenadores e desenvolvedores comprometida, organizada e que soube buscar o diálogo para a resolução de dúvidas e dificuldades.

Desenvolver um projeto de grande porte em equipe não é tarefa das mais fáceis, pois, em muitas ocasiões, a interação entre os membros do grupo é dificultada por motivos como incompatibilidade de horários e falhas de comunicação. A nossa equipe, porém, soube contornar esses e outros problemas, o que possibilitou a execução satisfatória de tudo aquilo que nos propusemos a fazer neste trabalho.

Assinalamos que:

a) um novo portal, ainda sem grandes elaborações estéticas, foi criado para o projeto DELPo, e deverá ser aperfeiçoado por futuros desenvolvedores;

b) todo o banco de dados do antigo DELPo foi recriado, e quase todas as informações foram migradas do banco antigo para o novo (alguns poucos dados, de importância secundária, deverão ser migrados futuramente, com a refatoração completa do projeto);

c) um mecanismo de autenticação similar ao que já existia no portal antigo foi bem desenvolvido e deverá ser complementado conforme a necessidade da equipe de coordenadores do DELPo;

d) a refatoração, em linguagem Ruby, de grande parte do complexo programa Moedor foi totalmente realizada, restando apenas acrescentar ao seu módulo *analise,* já desenvolvido, a função responsável por adicionar definitivamente ao banco de dados os resultados do processo de moagem de uma obra.

Nessa reelaboração do projeto DELPo, utilizamos ferramentas modernas, eficientes e de aprendizado relativamente fácil; também procuramos escrever códigos bem documentados e em linguagem clara. Através disso, objetivamos construir um sistema consistente, de fácil manutenção e que, além disso, pudesse ser incrementado e aperfeiçoado com relativa simplicidade, a fim de facilitar a atuação daqueles que desejem trabalhar na complementação do projeto que aqui desenvolvemos.

Referências Bibliográficas

TERRA, Ernani. **Curso Prático de Gramática**. 5. ed. São Paulo: Scipione, 2007.

CAELUM. **Desenvolvimento Ágil para Web com Ruby on Rails 4**. Curso RR-71. Disponível em: https://www.caelum.com.br/download/caelum-ruby-on-rails-rr71.pdf.

VIARO, Mário et al. **Manual do NEHiLP**. Disponível em: <http://www.usp.br/nehilp/infos/manual.pdf> . Acesso em: 23 out 2018.

VIARO, Mário. **O Dicionário Etimológico da Língua Portuguesa (DELPo): conceitos de metalema, hemilema, hiperlema e ultralema**. Disponível em: <http://siba-ese.unisalento.it/index.php/dvaf/article/view/17775>. Acesso em: 10 nov 2018.

SALVIANO, Bárbara Neves. **O uso do dicionário de língua como instrumento didático no ensino de Língua Portuguesa para alunos surdos: em busca de um bilinguismo funcional**. Disponível em: http://www.bibliotecadigital.ufmg.br/dspace/bitstream/handle/1843/MGSS-9LZNZ9/dissertacao2.pdf?sequence=1. Acesso em: 26 jan 2019.

PONTES, Antônio Luciano et al. **Perspectivas em Lexicografia e Terminologia.** 1. ed. Fortaleza: EdUECE, 2018.