

DELPO — DICIONÁRIO ETIMOLÓGICO DA LÍNGUA PORTUGUESA

Andre Luiz Abdalla Silveira (8030353)

Marco Dimas Gubitoso (Orientador)

Universidade de São Paulo — Instituto de Matemática e Estatística

Do que se trata?

O DELPo se trata de um portal onde pesquisadores podem investigar as origens de um verbete, podendo conhecer sua evolução temporal. O DELPo é um projeto do Nucleo de apoio à pesquisa em Etimologia e História da Língua Portuguesa (NEHiLP) criado pelo professor Mario Viario em 2012.

Além do **moedor** que é a funcionalidade principal do sistema, há também o multiplasmador que é toma uma palavra em latim vulgar e presume sua evolução em etapas até se tornar familiar ao que observamos nos dias de hoje. Há também ferramentas que permitem observar a frequência de determinada ocorrência ao passar dos anos.

Objetivos

- Fazer um sistema de duas camadas (interior e exterior) que provesse as funcionalidades já existentes e que pudesse escalar o bastante para confortar outras tantas
 - A camada interior (Back end) é onde se encontra a mior parte das regras de negócio — implementada em Ruby on Rails
 - A camada externa (Front end) é a de apresentação. É a interface de comunicação entre o usuário e a camada interior
- Refatorar o código do moedor, removendo os “cheiros” (problemas) do código
- Usar padrões de design na medida do possível
- Garantir que o código resultante seja o melhor possível para que quaisquer desenvolvedores possam escalar as duas camadas da aplicação

Back End – Ruby on Rails

Ruby on Rails é um framework em Ruby. Não se trata de algo revolucionário, visto que usa algumas ferramentas já validadas como o *ActiveRecord* (uma biblioteca que transforma entidades e relacionamentos em objetos), e o paradigma MVC (model - view - controller). Provavelmete o que possui o maior potencil é a possibilidade de criar uma aplicação com operações CRUD (create - read- update - delete) com relativamente pouco esforço

A aplicação feita em Ruby on Rails era a única camada do DELPo até o andamento deste projeto. No começo do ano, haviam controladores, modelos, mas não havia testes e as visualizações não apresentavam estilização alguma.

A primeira coisa a fazer foi conhecer a implementação atual para saber como tudo funcionava. Tudo funcionava, mas algumas refatorações como a da figura abaixo eram necessárias. Os dois trechos de código fazem a mesma coisa, a diferença é que na segunda imagem, o código está mais organizado

```
# recebe uma string de data e retorna para data no tipo aaaa-mm-dd
def tratadata (dat)
  # remove o c da incerteza
  dat.gsub!(/c/, '')
  #data completa
  if match = dat.match(/(\d+)(\d+)(\d+)(\d+)/)
    s1, s2, s3 = match.captures
    return s3 + "-" + s2 + "-" + s1
  elsif match = dat.match(/^(?!(1,2))(\d+)/)
    # sem dia, 05/1811, por exemplo
    s1, s2 = match.captures
    return ultimosdia(s1.to_s, s2.to_s)
  elsif match = dat.match(/(\d+)(\d+)/)
    #2000-2001, por exemplo
    s1 = match.captures
    return s1 + "-12-31"
  #XV-XVII, por exemplo
  elsif match = dat.match(/(X(0,3)(IV|IX|IV|V|I(0,3)))-(X(0,3)(IV|IX|IV|V|I(0,3)))/)
    s1, s2, s3 = match.captures
    ans = trataNumeral(s1) > trataNumeral(s3) ? trataNumeral(s1) : trataNumeral(s3)
    return ans + "-01-01"
  #2000 por exemplo
  elsif match = dat.match(/(0-9)(1,4)/)
    return dat + "-12-31"
  #XVII, por exemplo
  else
    return trataNumeral(dat).to_s + "-01-01" #1200-01-01 é o último dia do século XII
  end
end
```

```
def ultimo_dia_do_seculo (n)
  Date.new(RomanNumerals.to_decimal(n) * 100, -1, -1)
end

def eh_numerico? (ano)
  Integer(ano) != nil rescue false
end

def ajusta_data_com_dois_elementos (elemento_1, elemento_2)
  return ultimo_dia_do_seculo(elemento_2) unless eh_numerico?(elemento_2)
  begin
    data = Date.new(elemento_1.to_s, elemento_2.to_s, -1)
  rescue ArgumentError
    data = Date.new(elemento_2.to_s, -1, -1)
  ensure
    return data
  end
end

def ajusta_data_com_um_elemento(elemento_1)
  return ultimo_dia_do_seculo(elemento_1) unless eh_numerico?(elemento_1)
  Date.new(elemento_1.to_s, -1, -1)
end

def remove_incerteza_data
  data.gsub!(/c/, '')
end

def ajusta_data_data
  remove_incerteza(data)
  data_dividida = data.split('-')
  if data_dividida.length == 3
    data = Date.new(data_dividida[0].to_s, data_dividida[1].to_s, data_dividida[2].to_s)
  elsif data_dividida.length == 2
    ajusta_data_com_dois_elementos(data_dividida[0], data_dividida[1])
  else
    ajusta_data_com_um_elemento(data_dividida[0])
  end
end
```

Além disso, decidido por usar outro framework para fazer as visualizações, transformei a aplicação Rails em uma API (Application Program Interface) para aproveitar tudo o que o Rails tem a oferecer. A partir dessa decisão, e de necessidades consequentes, gemas foram adotadas e abandonadas, mantendo as funcionalidades, melhorando a qualidade do código (não estava ruim, mas alguns detalhes que passaram batido como aproveitar-se do ActiveRecord foram reparados).

Front End – Nuxt.js

Nuxt.js é um framework construído sobre outro: o Vue.js, que pode ser usado pra outras finalidades além da construção de páginas web. De uma forma similar ao Rails, que junta o que há de mais relevante no universo Ruby, o Nuxt é uma aplicação em Vue, só que com plug-ins essenciais como Vue Router (controla as rotas) e Vuex (onde ficam as “variáveis globais”) já vêm imbutidos na aplicação de modo que não se faz necessário lidar com esse e tantos outros detalhes. Além disso, Nuxt também otimiza o carregamento dos elementos da página, possibilitando uma melhor experiência de usuário.

O motivo principal pelo que se decidiu pelo uso do Nuxt ao invés do sistema nativo do Rails é o fato do último ser menos flexível em seu desenvolvimento. O desenvolvimento em Rails prefere convenções ao invés de configurações. Enquanto isso poupa algum trabalho a configurar suas preferências, por outro lado, prende o desenvolvedor a fazer as coisas do modo que “reza a cartilha” do framework, podendo muitas vezes dar terríveis dores de cabeça em quem se atreve a descomprir as convenções. Abaixo, a página principal da plataforma.



Reflexão Importante

Durante o desenvolvimento dessa aplicação, aprendi muitas coisas, e a principal é como equilibrar o ímpeto quase perfeccionista de fazer sempre o melhor código possível. Houveram ocaisões em que me encontrei bloqueado a pensar como fazer determinado método ou componente da melhor forma possível. Por um lado, permitiu-me fazer um código com poucas repetições, mais escalável, legível e compreensível. Por outro, fez-me perder dias, até mesmo semanas em que poderia estar fazendo mais coisas. O aprendizado que está se consolidando é atigir um equilíbrio entre a vontade de fazer entregar um produto viável ao cliente e um código quase impecável.

Bibliografia

- O dicionário etimológico da língua portuguesa (delpo): conceitos de metalema, hemilema, hiperlema e ultralema — Mário Eduardo Viaro — 2017
- Boas práticas — Daniel Schmitz — 2019
- Refatoração do Projeto Delpo — Adriano Tet-suaki Ogawa Santin, Luiz Fernando Antonelli Galati e Mauricio Luiz Abreu Cardoso — 2018
- O uso do dicionário de língua como instrumento didático no ensino de língua portuguesa para alunos surdos: em busca de um bilinguismo funcional — Barbara Neves Salviano — 2014
- Clean code — Wojtek Lukaszuk — 2018
- Ruby Cookbook — Lucas Carlson e Leonard Richardson — 2009
- Ruby on Rails Guides
- Vue.js Docs
- Nuxt.JS Guide
- Jest Homepage
- Manual do NEHiLP