Desenvolvimento de Interface em Python/Django para o NEHiLP

Fabio Brzostek Muller
Orientador: Prof. Dr. Marco Dimas Gubitoso
Bacharelado em Ciência da Computação
Instituto de Matemática e Estatística - Universidade de São Paulo (IME-USP)
fabio.muller@usp.br

1. Introdução

O Núcleo de Apoio à Pesquisa em Etimologia e História da Língua Portuguesa (NEHiLP), vinculado à Faculdade de Filosofia, Letras e Ciências Humanas da Universidade de São Paulo (FFLCH-USP), é um grupo que visa à divulgação das pesquisas acadêmicas brasileiras sobre Linguística Histórica, Filologia e Etimologia e tem como um dos principais projetos a criação do primeiro Dicionário Etimológico da Língua Portuguesa (DELPo).

Nos últimos anos, um sistema web foi desenvolvido para auxiliar os pesquisadores neste projeto: por meio desta interface, é possível inserir textos e informações relacionadas e consultar e modificar estas informações, armazenadas num banco de dados. O sistema, com código em PHP e Perl, está em uma fase estável, com as partes principais implementadas e funcionando. No entanto, ele se beneficiaria muito de uma reformulação, já que suas partes foram feitas por pessoas diversas e em vários períodos de tempo, misturando estilos de programação e ideias distintas quanto à organização.

O objetivo deste trabalho foi iniciar um processo de reescrita do sistema do NEHiLP, facilitando a manutenção e o desenvolvimento de novas funções no futuro, e assim contribuindo para a evolução do projeto. Para a nova versão, escolheu-se usar o *framework* Django, em Python, uma tecnologia bastante moderna e popular, com boas funções e muitos recursos disponíveis. Para fazer a adaptação, foi decidido primeiro criar os modelos a partir das tabelas do banco de dados, depois adaptar as páginas administrativas e de conteúdo e, por fim, se houvesse tempo, reescrever alguns dos programas e páginas restantes.

2. Organização e Estrutura

Estudando a estrutura do sistema existente, foi possível perceber uma divisão relativamente clara em algumas categorias: as páginas administrativas (perfil, configurações etc.), as de conteúdo estático e as "principais" (com os programas de busca, inserção e modificação dos dados).

No Django, um projeto ou um site costuma ser composto por *apps*, que são aplicações independentes, com um propósito definido. Considerando isso, decidiu-se criar, na versão nova, *apps* representando estas categorias.

Foram criados quatro *apps*, já que a categoria "principal" teve a parte relacionada a obras, autores, editoras etc. separada do resto. A divisão ocorreu por dois motivos: em primeiro lugar, as tabelas do banco representando estes conceitos formavam um subconjunto quase fechado, ideal para a primeira etapa de implementação do trabalho, um teste menos complexo de criação de modelos/páginas em Django. Além disso, um dos principais programas do projeto, o Moedor (cuja adaptação não fez parte deste trabalho), tem uma relação importante com este conjunto; como sua implementação será bastante complexa, faz sentido ele ficar fora do *app* "principal", evitando que este cresça demais.

Os *apps*, que são diretórios no projeto, têm uma estrutura similar, cujas partes principais são: os modelos (classes representando as tabelas do banco e suas propriedades), as *views* (funções que recebem requisições e retornam alguma resposta), as URLs do *app*, as *templates* (arquivos HTML que podem ter também alguns comandos especiais definidos pelo Django) e arquivos estáticos (JavaScript, CSS). O uso deste padrão na versão nova marcou uma grande melhora na organização dos arquivos em relação ao sistema existente, no qual eles ficam espalhados por diversos diretórios, o que faz com que seja difícil achar programas ou funções específicas. O esquema abaixo demonstra parte da nova estrutura.

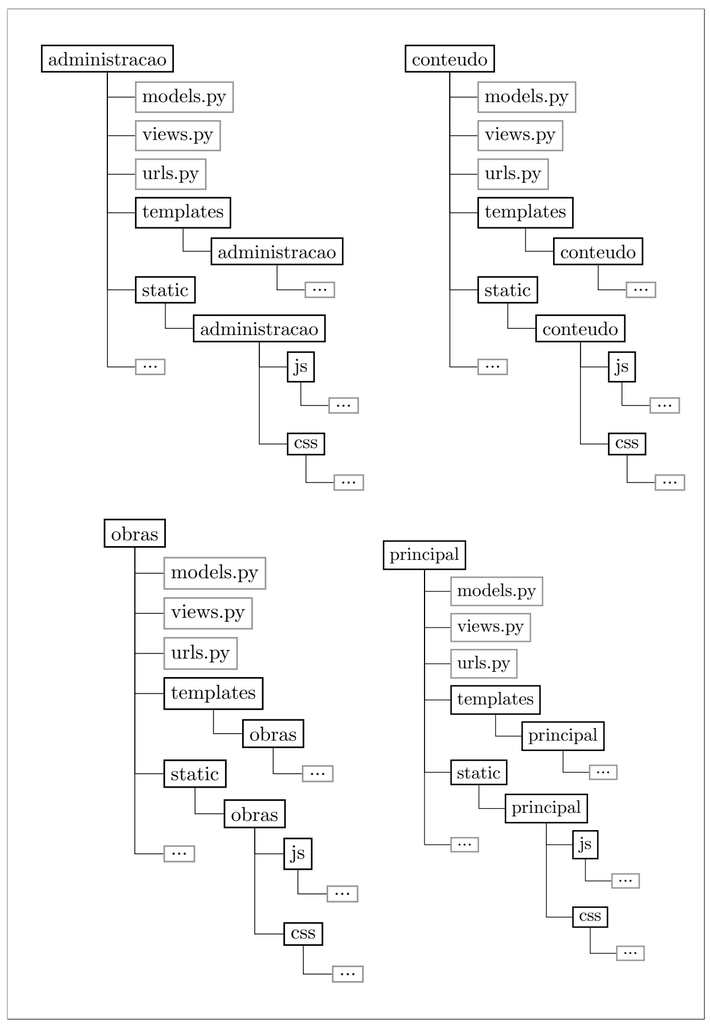

Figura 1: *Estrutura simplificada dos apps que compõem o sistema em Django*

3. Modelos, Páginas e Programas

O primeiro passo da implementação do novo sistema foi a criação dos modelos, as classes contendo todas as informações sobre cada tabela do banco. Neste processo, foram descobertas algumas inconsistências no banco de dados, a maioria das quais pôde ser corrigida direto no sistema já existente.

Após os modelos, foram criadas algumas páginas voltadas para testar se a integração com o banco estava funcionando corretamente. Depois disso, foram implementadas as páginas e funções administrativas e de conteúdo. Por fim, algumas das páginas de busca e de edição e inserção foram adaptadas.

O uso do Django permitiu que o código de vários programas fosse bastante simplificado por meio da utilização de funções presentes no *framework* e da sintaxe concisa de Python. As imagens a seguir exemplificam algumas das diferenças, mostrando versões simplificadas de um pedaço do código antigo e o código equivalente da versão nova.

Além de as consultas no Django serem significativamente mais curtas, já que usam o ORM do *framework* em vez de SQL puro, código Python foi usado no exemplo para processar de forma simples os resultados, evitando consultas extras ao banco e, assim, otimizando o processo.

PHP

```php
/* Para contar o total */
$sql_count = "SELECT count(*)
              FROM ocorrencia, contexto, obra
              WHERE ocorrencia.contexto_id = contexto.id
              AND contexto.obra_id = obra.id
              AND UPPER(ocorrencia.ocorrencia) LIKE UPPER('".$ocorrencia."')
              collate utf8_bin";
$result = mysql_query($sql_count);
$row = mysql_fetch_array($result);
$total = $row[0];

/* Pega resultados mostrados na página atual */
$sql = "SELECT ocorrencia.id, ocorrencia.ocorrencia, contexto.id, contexto.conteudo,
        obra.texto_titulo, year(obra.texto_data)
        FROM ocorrencia, contexto, obra
        WHERE ocorrencia.contexto_id = contexto.id
        AND contexto.obra_id = obra.id
        AND UPPER(ocorrencia.ocorrencia) LIKE UPPER('".$ocorrencia."') collate utf8_bin
        ORDER BY ocorrencia.ocorrencia, obra.texto_data ASC
        LIMIT " . $inicio . ", " . $itens_pag;

$result = mysql_query($sql);


/* Pega ano da ocorrência mais antiga e da mais recente */
$sqlg = "SELECT ocorrencia.id, ocorrencia.ocorrencia, contexto.id, contexto.conteudo,
         obra.texto_titulo, MIN(year(obra.texto_data)), MAX(year(obra.texto_data))
         FROM ocorrencia, contexto, obra
         WHERE ocorrencia.contexto_id = contexto.id
         AND contexto.obra_id = obra.id
         AND UPPER(ocorrencia.ocorrencia) LIKE UPPER('" . $ocorrencia . "') collate utf8_bin
         ORDER BY ocorrencia.ocorrencia, obra.texto_data ASC";
$resultg = mysql_query($sqlg);
$rowg = mysql_fetch_array($resultg);
$menordata = $rowg[5];
$maiordata = $rowg[6];

/* Calcula intervalo */
$intervalototal = $maiordata - $menordata;
$intervalolotes = $intervalototal / 10;

/* Pega quantidades de ocorrências por período */
$ocorrenciag = array();
for ($i = 1; $i <= 10; $i++) {
    $limitesuperior = $menordata + ($i) * $intervalolotes;
    $limiteinferior = $menordata + ($i - 1) * $intervalolotes;

    $sqlg2 = "SELECT ocorrencia.id, ocorrencia.ocorrencia, contexto.id,
              contexto.conteudo, obra.texto_titulo, year(obra.texto_data)
              FROM ocorrencia, contexto, obra
              WHERE ocorrencia.contexto_id = contexto.id
              AND contexto.obra_id = obra.id
              AND year(obra.texto_data) >= '$limiteinferior'
              AND year(obra.texto_data) <= '$limitesuperior'
              AND UPPER(ocorrencia.ocorrencia)LIKE UPPER('" . $ocorrencia . "')
              collate utf8_bin";
    $resultg2 = mysql_query($sqlg2);
    $nRowsg2 = mysql_num_rows($resultg2);

    $ocorrenciag[$i - 1] = $nRowsg2;
}
```

Figura 2: *Código (simplificado) em PHP, parte do sistema existente*

Python/Django

```python
# Pega todos os resultados
ocorrencias = Ocorrencia.objects.filter(
    ocorrencia__iexact=ocorrencia
).select_related('contexto__obra', 'variante__flexao__acepcao').order_by(
    'ocorrencia', 'contexto__obra__texto_data')

total = len(ocorrencias)

# Pega resultados mostrados na página atual
resultados = ocorrencias[inicio:final]

# Ordena resultados por data
ocorrencias_ordem_data = ocorrencias.order_by('contexto__obra__texto_data')

# Pega ano da ocorrência mais antiga e da mais recente
ano_mais_antigo = ocorrencias_ordem_data[0].contexto.obra.texto_data.year
ano_mais_recente = ocorrencias_ordem_data[total - 1].contexto.obra.texto_data.year

# Calcula intervalo
intervalo = ano_mais_recente - ano_mais_antigo
intervalo /= 10

# Pega quantidade de ocorrências por data
ocorrencias_por_datas = []
for i in range(1, num_iteracoes + 1):
    menor_ano = round(ano_mais_antigo + (i - 1) * intervalo)
    maior_ano = round(ano_mais_antigo + i * intervalo)

    ocorrencias_periodo = len([
        ocorr for ocorr in ocorrencias_ordem_data
        if ocorr.contexto.obra.texto_data.year >= menor_ano
        and ocorr.contexto.obra.texto_data.year <= maior_ano
    ])
    ocorrencias_por_datas.append(ocorrencias_periodo)
```

Figura 3: *Código (simplificado) em Python, parte correspondente do sistema novo*

4. Conclusões

O resultado do trabalho foi bastante satisfatório. Criou-se uma base muito importante para a adaptação do sistema para Django, incluindo a criação dos modelos e a integração com o banco e a implementação de várias páginas.

Além disso, foi possível resolver alguns problemas da versão existente do sistema e propor e utilizar uma nova organização. Isto tudo deve facilitar consideravelmente o trabalho de manutenção e desenvolvimento e contribuir para a evolução do projeto.