Engenharia de Dados e Conhecimento

universidade de aveiro

Trabalho Prático 1

2017/2018

I WanTo Know

António Sérgio Oliveira Silva
76678 asergio@ua.pt

1. Introdução ao Tema

Ao procurar um tema para o projeto, o objetivo foi o de encontrar uma API que devolvesse respostas em XML e que fosse interessante o suficiente para a profundidade do tema.

Após uma longa pesquisa, a API do Wolfram|Alpha foi encontrada.

O serviço web do Wolfram|Alpha é globalmente conhecida principalmente por estudantes de engenharia pela sua capacidade de realizar funções matemáticas. Porém, neste projeto, foram exploradas as outras capacidades menos conhecidas do Wolfram|Alpha, neste caso, como o tempo foi limitado, a aplicação inclui cinco escolhas para o utilizador:

- **“How much time has passed since my birthday”**
    - Devolve número de dias/semanas/anos desde dada data.

- **“What time is it in”**
    - Devolve horas actuais numa dada localização.

- **“Who was born in”**
    - Devolve pessoas notáveis que nasceram em dada data.

- **“How many calories on”**
    - Devolve número de calorias num dado alimento.

- **“How’s the weather in”**
    - Devolve temperatura atual numa dada localização.

2. Dados e suas fontes

Nesta aplicação, existem duas fontes de dados.

1. **Wolfram|Alpha API**

Como referido anteriormente, é usada a API do Wolfram|Alpha para adquirir o output do pedido realizado pelo utilizador.

Esta API é relativamente simples, os parâmetros necessários para o pedido são apenas a APPID (adquirido após registo) e INPUT (idêntico ao que escreveríamos na caixa de texto no website).

Exemplo de uma API call para obter as horas em Moscow:

```
api.wolframalpha.com/v2/query?appid=JX8868-T9QE9WHQTJ&input=time+in+moscow
```

A API devolve dados em XML.

Saber mais sobre a API

2. **Wolfram|Alpha Reddit Feed**

A aplicação também disponibiliza um feed de notícias relacionadas com o Wolfram|Alpha.

Este feed é obtido a partir do Atom Feed do Reddit, mais especificamente, do Sub-reddit “Wolfram|Alpha”.

Versão Web:
https://www.reddit.com/r/wolframalpha/

Atom Feed:
https://www.reddit.com/r/wolframalpha.xml

3. Esquema dos dados (XMLSchema)

Foi usado XMLSchema para validação do XML devolvido pela API.

Exemplo de um XML devolvido pela API:

```
▼<queryresult success="true" error="false" numpods="4" datatypes="CalendarEvent,City" timedout="" timedoutpods="" related="http://www4b.wolframalpha.com/api/v2/relatedQueries.jsp?id=MSPa359912bbb8gdg6d66d7a0000611hg8c4i6702ha3
  ▶<pod title="Input interpretation" scanner="Identity" id="Input" position="100" error="false" numsubpods="1">..
  ▶<pod title="Result" scanner="Identity" id="Result" position="200" error="false" numsubpods="1" primary="true">
  ▶<pod title="Time offset from WET" scanner="Date" id="TimeOffsets" position="300" error="false" numsubpods="1">
  ▶<pod title="Clocks" scanner="Date" id="ClockImages" position="400" error="false" numsubpods="1">...</pod>
  ▶<assumptions count="1">...</assumptions>
  ▶<sources count="1">...</sources>
  </queryresult>
```

O formato é simples. Os vários outputs vêm em elementos com o nome “pod”, apesar que, o output desejado encontra-se no elemento “pod” que contém o id ‘Result’. Então é seguro assumir que o XML é válido se conter pelo menos dois elementos ‘pod’ e que o atributo ‘success’ do elemento ‘queryresult’ tenha o valor ‘true’.

```
<xs:element name="queryresult">
  <xs:complexType>

    <xs:sequence>

      <xs:element name="pod" minOccurs="2" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
              <xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute type="xs:string" name="title" use="optional"/>
          <xs:attribute type="xs:string" name="scanner" use="optional"/>
          <xs:attribute type="xs:string" name="id" use="optional"/>
          <xs:attribute type="xs:short" name="position" use="optional"/>
          <xs:attribute type="xs:string" name="error" use="optional"/>
          <xs:attribute type="xs:byte" name="numsubpods" use="optional"/>
          <xs:attribute type="xs:string" name="primary" use="optional"/>
        </xs:complexType>
      </xs:element>

      <xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>

    </xs:sequence>

    <xs:attribute type="xs:string" name="success" fixed="true"/>
```

A validação é feita com recurso à criação de um parser. Esse parser é depois usado para a leitura do XML pelo que falhará se não for válido.

```
# Returns parser for XML schema validation
def get_schema_parser(action):
	schema_root = eTree.parse('app/static/schemas/' + action + '.xsd')
	xsd_file = eTree.XMLSchema(schema_root)
	parser = eTree.XMLParser(schema = xsd_file)
	return parser
```

4. Transformação sobre os dados

Depois de leitura e validação do output, esta é transformada de maneira a ser apresentada ao utilizador na página HTML usando o correspondente ficheiro XSL.

Trata-se, em geral, de transformações simples como listas (
) e aplicação de classes.

Também é usada estas transformações para apresentação do Feed Atom do Reddit.

```
<xsl:template match="/">
        <ul>
            <xsl:for-each select=".//atom:entry">
                <xsl:variable name="HREF">
                    <xsl:value-ofselect="atom:link/@href"/>
                </xsl:variable>
                <li class="feed_content">
                    <a href= "{$HREF}"> <xsl:value-of select="atom:title"/> </a>
                </li>
            </xsl:for-each>
        </ul>
    </xsl:template>
```

5. Operações sobre os dados

Pela procura de uma forma de melhor explorar o XQuery e de dar uma maior funcionalidade à aplicação, todos os inputs dados pelo o utilizador que resultaram num output válido são armazenados na base de dados.

O objetivo de tal é fornecer ao utilizador uma sugestão de input (neste caso, a mais popular).

```
<entrie>
  <title>What time is it in</title>
  <action>time_in</action>
  <user_inputs>
    <user_input times="2"> Moscow </user_input>
    <user_input times="10">Spain</user_input>
    <user_input times="1">Russia</user_input>
    <user_input times="4">Moscow</user_input>
  </user_inputs>
</entrie>
```

O exemplo mostra um elemento “entrie”, a base de dados baseia-se em conjunto de elementos do mesmo gênero. Neste caso, é o elemento que contém a funcionalidade “What time is it in”, dentro deste elemento podemos também encontrar o elemento “action”, que é usado para gerar a action nos forms correspondentes, e o elemento “user_inputs” que contém os vários inputs já fornecidos pelo utilizador.

Cada elemento filho do “user_inputs” contém um atributo “times” que corresponde a quantas vezes esse input for fornecido.

O trabalho da aplicação é, então, quando recebe um input, verificar se este já existe. Caso exista incrementa o valor de times, caso nao exista adiciona um elemento “user_input” novo.

Na imagem em baixo é possível ver a query que corresponde a tal. Baseia-se em `if exists(...) then (replace ...) else (insert node ...)`

```python
def store_user_input(action, user_input):
    session = get_db_session()

    if session:
        # Location of user_inputs for this action
        location = "collection('entries')//entrie/action[text()=\""+ action +"\"]/../user_inputs"
        # Condition to check if input already exists
        exists = "(exists(" + location + "/user_input[text()=\""+ user_input +"\"]))"
        # If already exists it will increments his 'times' value, else will create a new user_input
        query = "xquery if " + exists + " then " \
                "(replace value of node (" + location + "/user_input[text()=\""+ user_input +"\"]/@times
                "(insert node (<user_input times=\"1\">" + user_input + "</user_input>) into  " + locati

        result = session.execute(query)

        session.close()
```

6. Funcionalidades da Aplicação

Na sua versão actual, a aplicação não apresenta um grande número de funcionalidade. Tal deve-se simplesmente ao tempo limitado para o desenvolver. Contudo, é interessante notar que a adição de novas funcionalidades torna-se num trabalho facilitado com as tecnologias usadas.

Com esta aplicação é capaz de:

- Conhecer as últimas notícias sobre o Wolfram
- Saber quanto dias passaram desde determinada data
- Saber que pessoas notáveis nasceram em determinada data
- Saber as horas actuais em dado local
- Saber que calorias se encontram em dado alimento
- Saber a temperatura atual em dado local

Notar também que a aplicação armazena o input do utilizador para mais tarde o usar como sugestão.

7. Conclusões

Num mundo controlado por dados como a Internet, é necessário conhecer as tecnologias disponíveis para o controlo da mesma.

A investigação das tecnologias deste projeto deu a concluir que muito trabalho pode ser facilitado na parte de framework com o uso correto de operação e validação de dados.

Apesar das madrugadas com trinta tabs do browser abertas para tentar entender o erro de syntax no xquery que era gerado por simplesmente a falta de um espaço, foi um projeto que deu gosto em desenvolver e, provavelmente, futuramente aprofundar.

8. Configuração para Executar a Aplicação

A aplicação é dependente da base de dados!

É necessário executar o BaseXServer e ter nele disponível a base de dados “entries” (ficheiro para importar em `/app/static/database/entries.xml`)

Para importar a base de dados (na pasta app/static/database/):

```
java -cp BaseX867.jar org.basex.BaseXClient -n 127.0.0.1 -c"CREATE DB entries entries.xml"
```

Para iniciar o BaseXServer (na pasta app/static/database/):

```
java -cp BaseX867.jar org.basex.BaseXServer -n 127.0.0.1
```

Quanto a packages python, as únicas não default usadas são “django” para uso da framework e “lxml” para operações com o ficheiros xml. Caso não as possua, facilmente as pode descarregar com:

```
pip3 install lxml django
```