

Departamento de Engenharia Informática e de Sistemas
Instituto Superior de Engenharia de Coimbra

Licenciatura em Engenharia Informática

Integração de Dados

2021/2022



Trabalho prático

Autores:

Maria José Gonçalves Marcos – 2018018386

Tiago Alexandre Pais Dias - 2019126344

Índice

Introdução.....	3
Estrutura do projeto.....	3
Análise das fontes de dados.....	4
<i>Wrappers</i> Implementados	5
Interface.....	9
Menus.....	9
Exemplos	10
Ficheiros fornecidos	12
Modelo <i>XML</i>	12
Validação do modelo.....	14
Ficheiro DTD	14
Ficheiro XSD.....	15
Pesquisas <i>XPATH</i>	17
Ficheiros <i>XSLT/XQuery</i>	19
XSLT	19
Xquery.....	21
Conclusão.....	23

Introdução

Com a realização deste trabalho pretende-se criar um programa em Java, composto por vários *wrappers* que obtenham dados de fontes heterogéneas, distribuídas e autónomas e possibilitem ao utilizador a visualização dos dados de forma integrada. A seguinte aplicação integradora apresenta uma visão unificada de informações relativas a cidades de diferentes países.

Estrutura do projeto

O projeto é constituído pelos vários ficheiros:

- TpCidadesDoMundo.java (main);
- Cidades.java;
- Wrappers.java;
- ModeloXML.java;
- HttpRequestFunctions.java;
- JDOMFunctions_Validar.java;
- XMLJDomFunctions.java;
- XPathFunctions.java;
- JDOMFunctions_XSLT;
- SaxonFunctions_XQuery;
- Interface.form;
- Interface.java;

E os ficheiros auxiliares:

- cidade.txt;
- cidade2.txt;
- cidades.dtd;
- cidades.xml;
- cidades.xsd;
- cidades.txt
- CidadesPais.xql
- detalhesCidade.xml
- ficheiro_cidades.txt
- fotosMonumentos.html
- fuso.xql
- linguagens.xql
- linguagensCidade.html
- ficheiro_cidades.txt;
- pais.txt;
- transf1.xsl;
- transf2.xsl;
- transf3.xsl;
- cidadesEspanha.txt;
- imagensBandeiras.html;

- listagemCidadesPopulosas.xml;
- monumentos.xql
- pedido.xml

Análise das fontes de dados

Inicialmente foi analisada a distribuição da informação das cidades pelos websites. A decisão foi feita com base na existência, ou inexistência, de determinada informação e na simplicidade das expressões regulares.

A tabela ilustrada abaixo, mostra o website escolhido para a retirada de cada um dos atributos.

	Wikipédia	Db-city
Nome da cidade		
País a que pertence		
Indicação se a cidade é a capital do país		x
Link para a imagem da bandeira do país		x
Língua(s) oficial(ais) do país		x
Link para a imagem da bandeira da cidade	x	
Links para as imagens de monumentos/ <i>landmarks</i> da cidade	x	
Área da cidade (valor numérico em km2)		x
Nº de habitantes da cidade		x
Densidade populacional da cidade (nº de habitantes por km2)		x
Código Postal da cidade		x
Presidente da Câmara da cidade		x
Latitude e Longitude da cidade		x
Altitude da cidade em metros		x
Clima da cidade		x
Fuso Horário da cidade		x
Website da cidade		x
Cidades geminadas		x

Wrappers Implementados

String obterLinkdbCity(String cidade,String pais)

A função obterLinkdbCity recebe o país e o nome da cidade e devolve o link para a página principal da cidade indicada. Para tal, primeiro junta a String do país ao link da página DbCity e, com recurso ao httpRequest, obtém a página principal com a informação daquele país, guardando o respetivo source num ficheiro txt ("pais.txt"). De seguida pesquisa nesse source o link da cidade indicada recorrendo à expressão regular:

```
String er ="href=\"/[A-Za-z-ç]+--[A-Za-z-Ø]+[^\"]+\" title=\"\"+cidade+"\">"+cidade+"</a>";
```

String obterLinkWikipedia(String cidade)

A função obterLinkWikipedia recebe o nome de uma cidade e junta essa String ao link da página da Wikipédia e, com recurso ao httpRequest, obtendo a página principal da cidade indicada e guardando o seu respetivo source num ficheiro txt ("cidade.txt"). As próximas funções fazem as suas pesquisas no ficheiro txt acima mencionado. Para tal recorre à expressão regular:

```
String er ="\"wgPageName\"::\"([A-Za-z- _]+)\"";
```

boolean obterCapital(String cidade,String pais)

A função acima devolve *true* se a cidade indicada for a capital, ou *false* caso não seja. A informação é retirada do *dbCity*, com o auxílio da função *obterLinkdbCity*, guardada no ficheiro ("pais.txt"). Para tal recorre ao uso da expressão regular:

```
String er =">"+cidade+"<\\V> \\(Capital\\),";
```

String obterBandeiraPais(String cidade, String pais)

A função acima devolve uma *string* com o link do *source* da imagem da bandeira do país indicado. A informação é retirada do *dbCity*, com o auxílio da função *obterLinkdbCity*, e posteriormente com um novo httpRequest ao link devolvido por essa função, e guardando o respetivo source no ficheiro ("cidade2.txt"). Para tal recorre ao uso da expressão regular:

```
String er ="<img src=\"([^\"]+)\" alt=\"Bandeira [^\"]+\" />";
```

String obterBandeiraCidade(String cidade)

A função acima devolve uma *string* com o link do *source* da imagem da bandeira da cidade indicada. A informação é retirada da *Wikipédia*, com o auxílio da função *obtemLinkWikipedia* e guardada no ficheiro ("cidade.txt"). Para tal recorre ao uso da expressão regular:

```
String er ="<img alt=\"Bandeira de "+cidade+"[^"]*" src=\"([^\"]+)\"";
```

String obterPresidente(String cidade, String pais)

A função acima devolve uma *string* com o nome do presidente da cidade indicada. A informação é retirada do *dbCity*, com o auxílio da função *obtemLinkdbCity*, e posteriormente com um novo *httpRequest* ao link devolvido por essa função, e guardando o respetivo *source* no ficheiro ("cidade2.txt"). Para tal recorre ao uso da expressão regular:

```
String er ="<th>Presidente da Câmara "+cidade+"</th><td>([^\"]+)</td>";
```

String obterClima(String cidade, String pais)

A função acima devolve uma *string* com o clima da cidade indicada. A informação é retirada do *dbCity*, com o auxílio da função *obtemLinkdbCity*, e posteriormente com um novo *httpRequest* ao link devolvido por essa função, e guardando o respetivo *source* no ficheiro ("cidade2.txt"). Para tal recorre ao uso da expressão regular:

```
String er ="<th>Clima "+cidade+"</th><td>([^\"]+)</td>";
```

String obterFuso(String cidade, String pais)

A função acima devolve uma *string* com o fuso horário da cidade indicada. A informação é retirada do *dbCity*, com o auxílio da função *obtemLinkdbCity*, e posteriormente com um novo *httpRequest* ao link devolvido por essa função, e guardando o respetivo *source* no ficheiro ("cidade2.txt"). Para tal recorre ao uso da expressão regular:

```
String er ="<th>Fuso horário "+cidade+"</th><td><abbr  
title=\"([^\"]+)>\"([^\"]+)</abbr>([^\"]+)<br>";
```

String obterWebsite(String cidade, String pais)

A função acima devolve uma *string* com o *website* da cidade indicada. A informação é retirada do *dbCity*, com o auxílio da função *obtemLinkdbCity*, e posteriormente com um novo *httpRequest* ao link devolvido por essa função, e guardando o respetivo *source* no ficheiro ("cidade2.txt"). Para tal recorre ao uso da expressão regular:

```
String er ="<th>Sítio Web "+cidade+"</th><td><a class=\"url\" href=\"([^\"]+)\"";
```

double obterArea(String cidade, String pais)

A função acima devolve um *double* com a área, em km², da cidade indicada. A informação é retirada do *dbCity*, com o auxílio da função *obtemLinkdbCity*, e posteriormente com um novo *httpRequest* ao link devolvido por essa função, e guardando o respetivo *source* no ficheiro ("cidade2.txt"). Para tal recorre ao uso da expressão regular:

```
String er ="<th>Superfície "+cidade+"</th><td>^[^<br />]+([0-9]*.[0-9]+,[0-9]*)  
km²</td>";
```

double obterDensidade(String cidade,String pais)

A função acima devolve um *double* com a densidade populacional, nº de habitantes por km², da cidade indicada. A informação é retirada do *dbCity*, com o auxílio da função *obtemLinkdbCity*, e posteriormente com um novo *HttpRequest* ao link devolvido por essa função, e guardando o respetivo source no ficheiro ("cidade2.txt"). Para tal recorre ao uso da expressão regular:

```
String er ="<th>Densidade populacional "+cidade+"</th><td>([0-9]*.[0-9]+,[0-9]*)/km²</td>";
```

double obterLatitude(String cidade,String pais)

A função acima devolve um *double* com a latitude da cidade indicada. A informação é retirada do *dbCity*, com o auxílio da função *obtemLinkdbCity*, e posteriormente com um novo *HttpRequest* ao link devolvido por essa função, e guardando o respetivo source no ficheiro ("cidade2.txt"). Para tal recorre ao uso da expressão regular:

```
String er ="Latitude: <b class=\"latitude\">(-?[0-9]+.[0-9]+)</b>";
```

double obterLongitude(String cidade,String pais)

A função acima devolve um *double* com a longitude da cidade indicada. A informação é retirada do *dbCity*, com o auxílio da função *obtemLinkdbCity*, e posteriormente com um novo *HttpRequest* ao link devolvido por essa função, e guardando o respetivo source no ficheiro ("cidade2.txt"). Para tal recorre ao uso da expressão regular:

```
String er ="Longitude: <b class=\"longitude\">(-?[0-9]+.[0-9]+)</b>";
```

int obterNHabitantes(String cidade,String pais)

A função acima devolve um inteiro com o número de habitantes da cidade indicada. A informação é retirada do *dbCity*, com o auxílio da função *obtemLinkdbCity*, e posteriormente com um novo *HttpRequest* ao link devolvido por essa função, e guardando o respetivo source no ficheiro ("cidade2.txt"). Para tal recorre ao uso da expressão regular:

```
String er ="<th>Número de habitantes "+cidade+"</th><td>([0-9]*.[0-9]+.[0-9]*) habitantes</td>";
```

int obterCodigoPostal(String cidade,String pais)

A função acima devolve um inteiro com o código postal da cidade indicada. A informação é retirada do *dbCity*, com o auxílio da função *obtemLinkdbCity*, e posteriormente com um novo *HttpRequest* ao link devolvido por essa função, e guardando o respetivo source no ficheiro ("cidade2.txt"). Para tal recorre ao uso da expressão regular:

```
String er ="<th>Código postal "+cidade+"</th><td>([0-9]+)</td>";
```

int obterAltitude(String cidade,String pais)

A função acima devolve um inteiro com a altitude da cidade indicada. A informação é retirada do *dbCity*, com o auxílio da função *obtemLinkdbCity*, e posteriormente com um novo *HttpRequest* ao link devolvido por essa função, e guardando o respetivo source no ficheiro ("cidade2.txt"). Para tal recorre ao uso da expressão regular:

```
String er ="<th>Altitude "+cidade+"</th><td>([0-9]+) m</td>";
```

ArrayList obterLinguagens(String cidade,String pais)

A função procura todas as linguagens oficiais do país indicado e armazena-as numa *ArrayList* de *Strings*. A informação é retirada do *dbCity*, com o auxílio da função *obtemLinkdbCity*, e posteriormente com um novo *HttpRequest* ao link devolvido por essa função, e guardando o respetivo source no ficheiro ("cidade2.txt"). Para tal utiliza a seguinte expressão regular. Caso um país possua várias linguagens oficiais, a função separa-as utilizando a função *split("
")*.

```
String er ="<th>Língua oficial</th><td>([^\<]+<?[\^<]*<?[\^<]*)</td>";
```

ArrayList obterMonumentos(String cidade)

A função procura o *link* dos *sources* de imagens de monumentos (*landmarks*) da cidade indicada e armazena-os num *ArrayList* de *Strings*. A informação é retirada da *Wikipédia*, com o auxílio da função *obtemLinkWikipedia* e guardada no ficheiro ("cidade.txt"). Para tal utiliza as seguintes expressões regulares:

Caso a cidade seja Detroit, Porto, Munique, Lyon ou Oslo, pois estas cidades apenas possuem uma montagem com todos os monumentos, a expressão é:

```
String er ="<meta property=\"og:image\" content=\"([^\"]+)\"/>";
```

Para as restantes são apenas obtidos os *links* de seis monumentos, excluindo as imagens com extensão (".svg.png"), a expressão é:

```
String er ="<a href=\"([^\"]+)\" class=\"image\"><img alt=\"([^\"]*)\" src=\"([^\"]+)\" decoding=\"([^\"]+)\" ?w?i?d?t?h?=?\"?([^\"]*)\"? ?h?e?i?g?h?t?=?\"?([^\"]*)\"? srcset=\"([^\"]+)\" data-file-width=\"([^\"]+)\" data-file-height=\"([^\"]+)\" w?i?d?t?h?=?\"?([^\"]*)\"? ?h?e?i?g?h?t?=?\"?([^\"]*)\"?/></a>";
```

ArrayList obterCidadesGeminadas(String cidade,String pais)

A função procura todas as cidades geminadas da cidade indicada e armazena-as numa *ArrayList* de *Strings*. A informação é retirada do *dbCity*, com o auxílio da função *obtemLinkdbCity*, e posteriormente com um novo *HttpRequest* ao link devolvido por essa função, e guardando o respetivo source no ficheiro ("cidade2.txt"). Para tal utiliza a seguinte expressão regular:

```
String er ="class=\"img_drp\" /></a> <a href=\"([^\"]+)\" title=\"([^\"]+)\>[^\<]+</a>";
```

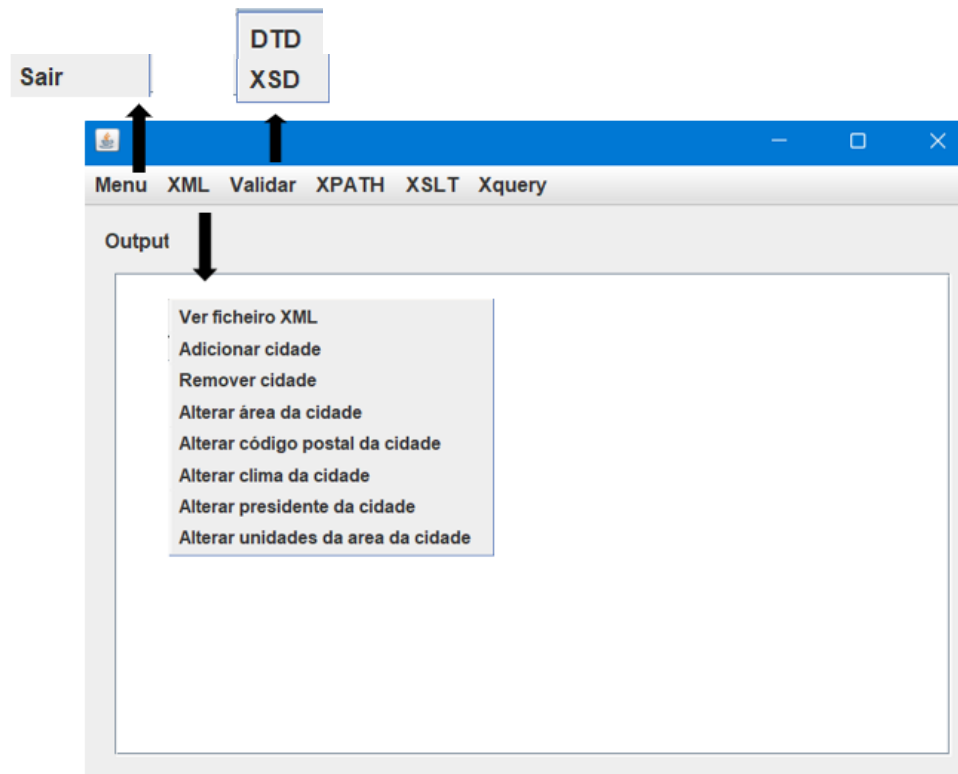

Interface

Menus

A aplicação construída para interagir com o utilizador encontra-se dividida pelos menus:

- ❖ Menu
 - Sair
- ❖ XML
 - Ver ficheiro XML;
 - Adicionar cidade;
 - Remover cidade;
 - Alterar área da cidade;
 - Alterar código postal da cidade;
 - Alterar clima da cidade;
 - Alterar presidente da cidade;
 - Alterar unidades da área da cidade;
- ❖ Validar
 - DTD;
 - XSD;
- ❖ XPATH
 - Pesquisar por nome;
 - Pesquisar por país;
 - Pesquisar por número de habitantes;
 - Pesquisar por clima;
 - Pesquisar por capitais;
 - ...
- ❖ XSLT
 - XML >-- HTML fotos das bandeiras
 - XML >-- TXT cidades de um país
 - XML >-- XML cidades mais populosas
- ❖ Xquery
 - HTML fotos dos monumentos
 - XML detalhes cidade
 - TXT cidades por fuso
 - HTML cidades por linguagem

Exemplos



Outros componentes:

Além dos menus, a aplicação desenvolvida recorre também a Janelas de Diálogo através das quais pede informação ao utilizador, Janelas de Informação/Erro para dar feedback ao utilizador.

Adiciona nova cidade

Nome da cidade:

Nome do país:

Adicionar

Remove cidade

Nome da cidade:

Remover


Altera area da cidade

Nome da cidade:

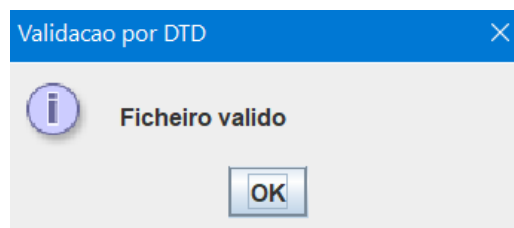
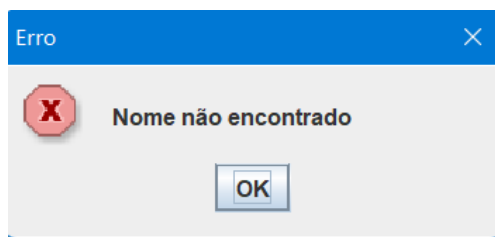
Nova área:

Alterar

Informação

 Cidade adicionada com sucesso

OK



Ficheiros fornecidos

Modelo XML

O documento XML segue a seguinte estrutura:

```
<catalogo>
  <cidade>
    <nome> </nome>
    <pais> </pais>
    <bandeirapais> </bandeirapais>
    <bandeiracidade> </bandeiracidade>
    <capital> </capital>
    <presidente> </presidente>
    <clima></clima>
    <fuso></fuso>
    <website> </website>
    <area uni=" "></area>
    <densidade uni=" "></densidade>
    <linguagens>
      <lingua> </lingua>
      ...
    </linguagens>
    <monumentos>
      <monumento> </monumento>
      ...
    </monumentos>
    <ciudadesgeminadas>
      <cidadegeminada> </cidadegeminada>
      ...
    </ciudadesgeminadas>
    <latitude> </latitude>
    <longitude> </longitude>
    <altitude></altitude>
    <nhabitantes></nhabitantes>
    <codigopostal></codigopostal>
  </cidade>
  ...
</catalogo>
```

No que toca à manipulação do ficheiro XML, implementamos diversas funcionalidades. Estas passam pela inserção de novas cidades, remoção de cidades e alteração de determinados atributos de uma cidade escolhida.

A inserção de cidades é feita através do item “Adicionar Cidade”. Este item abre uma janela em que é pedido o nome da cidade e o nome do país (estes devem ser inseridos corretamente e devem começar por um carácter maiúsculo). De seguida é invocada a função *criaCidade(String cidade_, String pais_)*, que vai executar os Wrappers acima

mencionados e devolver um objeto do tipo Cidades que é depois adicionado ao documento XML (“cidades.xml”).

A remoção de uma cidade é feita através do item “Remover Cidade”. Este item abre uma janela em que é pedido o nome da cidade ao utilizador. Com isto, a função *removeCidade(String nome, Document doc)* procura a cidade com o nome indicado por parâmetro e elimina-o do documento XML (indicado também como parâmetro).

Quanto à alteração dos atributos de uma cidade, é possível alterar a área, o código postal, o clima, o nome do presidente e ainda alterar as unidades da área. Para cada item, item abre uma janela em que é pedido o nome da cidade ao utilizador e o novo valor do atributo alterar. Com isto, é chamada a função respetiva ao atributo escolhido e é feita a alteração. As funções são:

alteraArea(String nome, double novaArea, Document doc)

alteraCodigoPostal(String nome, int codigoPostal, Document doc)

alteraClima(String nome, String clima, Document doc)

alteraPresidente(String nome, String presidente, Document doc)

alteraUniArea(String nome, String uniNova, Document doc)

Em todas as situações o utilizador recebe um alerta a indicar o sucesso ou insucesso de cada operação.

Validação do modelo

É possível através da aplicação efetuar a validação do documento XML por DTD e por XSD. Ao longo das aulas foi-nos fornecido o ficheiro JDOMFunctions_validar.java, composto pelas seguintes funções:

- Document validarDTD(String caminhoFicheiro), que executa a validação do documento XML usando DTD;
- Document validarXSD(String caminhoFicheiro), que executa a validação do documento XML usando XSD
- int validarDocumentoDTD(String xmlFile, String DTDFile), que atribui a validação DTD ao ficheiro XML, devolve 1 se o documento for válido por DTD, ou -1 caso contrário.
- int validarDocumentoXSD(String xmlFile, String XSDFile), que atribui a validação XSD ao ficheiro XML, devolve 1 se o documento for válido por XSD, ou -1 caso contrário.

Para o correto funcionamento destas funções é necessário o recurso à API JDOM e SAXON. Foram adicionadas duas bibliotecas da API SAXON, saxon9.jar e saxon-s9api.jar.

Ficheiro DTD

```
<!ELEMENT nome (#PCDATA)>
<!ELEMENT pais (#PCDATA)>
<!ELEMENT bandeirapais (#PCDATA)>
<!ELEMENT bandeiracidade (#PCDATA)>
<!ELEMENT capital (#PCDATA)>
<!ELEMENT presidente (#PCDATA)>
<!ELEMENT clima (#PCDATA)>
<!ELEMENT fuso (#PCDATA)>
<!ELEMENT website (#PCDATA)>
<!ELEMENT area (#PCDATA)>
<!ELEMENT densidade (#PCDATA)>
<!ELEMENT lingua (#PCDATA)>
<!ELEMENT monumento (#PCDATA)>
<!ELEMENT cidadegeminada (#PCDATA)>
<!ELEMENT linguagens (lingua*)>
<!ELEMENT monumentos (monumento*)>
<!ELEMENT cidadesgeminadas (cidadegeminada*)>
<!ELEMENT latitude (#PCDATA)>
<!ELEMENT longitude (#PCDATA)>
<!ELEMENT altitude (#PCDATA)>
<!ELEMENT nhabitantes (#PCDATA)>
<!ELEMENT codigopostal (#PCDATA)>
<!ELEMENT cidade (nome,pais,bandeirapais? ,bandeiracidade? ,capital, presidente?
,clima? ,fuso? ,website? ,area? ,densidade? ,linguagens* ,monumentos*
,cidadesgeminadas* ,latitude? ,longitude? ,altitude? ,nhabitantes?
,codigopostal?)>
<!ELEMENT catalogo (cidade)+>
<!ATTLIST area uni CDATA #REQUIRED>
<!ATTLIST densidade uni CDATA #REQUIRED>
<!ATTLIST catalogo xmlns:xsi CDATA #IMPLIED>
<!ATTLIST catalogo xsi:noNamespaceSchemaLocation CDATA #IMPLIED>
```

Ficheiro XSD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:attribute name="uni" type="xsd:string"/>

  <xsd:element name="nome" type="xsd:string"/>
  <xsd:element name="pais" type="xsd:string"/>
  <xsd:element name="bandeirapais" type="xsd:string"/>
  <xsd:element name="bandeiracidade" type="xsd:string"/>
  <xsd:element name="capital" type="xsd:string"/>
  <xsd:element name="presidente" type="xsd:string"/>
  <xsd:element name="clima" type="xsd:string"/>
  <xsd:element name="fuso" type="xsd:string"/>
  <xsd:element name="website" type="xsd:string"/>
  <xsd:element name="area">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="xsd:double">
          <xsd:attribute ref="uni" use="required"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="densidade">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="xsd:double">
          <xsd:attribute ref="uni" use="required"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="lingua" type="xsd:string"/>
  <xsd:element name="monumento" type="xsd:string"/>
  <xsd:element name="cidadegeminada" type="xsd:string"/>
  <xsd:element name="latitude" type="xsd:double"/>
  <xsd:element name="longitude" type="xsd:double"/>
  <xsd:element name="altitude" type="xsd:integer"/>
  <xsd:element name="nhabitantes" type="xsd:integer"/>
  <xsd:element name="codigopostal" type="xsd:integer"/>
  <xsd:element name="linguagens">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="lingua" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="monumentos">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="monumento" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```

        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="cidadesgeminadas">
    <xsd:complexType>
        <xsd:sequence>
<xsd:element ref="cidadegeminada" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="cidade">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="nome" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="pais" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="bandeirapais" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="bandeiracidade" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="capital" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="presidente" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="clima" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="fuso" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="website" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="area" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="densidade" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="linguagens" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="monumentos" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="cidadesgeminadas" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="latitude" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="longitude" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="altitude" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="nhabitantes" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="codigopostal" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="catalogo">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="cidade" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>

```


Pesquisas *XPATH*

As pesquisas com *XPath* pedem ao utilizador um valor a pesquisar e apresentam a lista com os resultados encontrados na janela de Output. Pesquisas realizadas:

Pesquisar por nome

É pedido o nome da cidade e a pesquisa mostra a informação relevante sobre ela, país a que pertence, se é capital (*true/false*), clima, fuso horário, *website*, área. Densidade populacional, número de habitantes, latitude, longitude, altitude, presidente e o código postal

```
xp = "//cidade[contains(nome, '" + jTextField14.getText() + "')] /pais | "
      + "//cidade[contains(nome, '" + jTextField14.getText() + "')] /capital | "
      + "//cidade[contains(nome, '" + jTextField14.getText() + "')] /clima | "
      + "//cidade[contains(nome, '" + jTextField14.getText() + "')] /fuso | "
      + "//cidade[contains(nome, '" + jTextField14.getText() + "')] /website | "
      + "//cidade[contains(nome, '" + jTextField14.getText() + "')] /area | "
      + "//cidade[contains(nome, '" + jTextField14.getText() + "')] /densidade | "
      + "//cidade[contains(nome, '" + jTextField14.getText() + "')] /nhabitantes | "
      + "//cidade[contains(nome, '" + jTextField14.getText() + "')] /latitude | "
      + "//cidade[contains(nome, '" + jTextField14.getText() + "')] /longitude | "
      + "//cidade[contains(nome, '" + jTextField14.getText() + "')] /altitude | "
      + "//cidade[contains(nome, '" + jTextField14.getText() + "')] /presidente | "
      + "//cidade[contains(nome, '" + jTextField14.getText() + "')] /codigopostal";
```

Pesquisar por país

É pedido o nome do país e a pesquisa mostra todas as cidades existentes no ficheiro pertencentes a esse país.

```
xp = "//cidade[contains(pais, '" + jTextField14.getText() + "')] /nome";
```

Pesquisar por nº de habitantes superior a

É pedido um número de habitantes e a pesquisa mostra todas as cidades existentes no ficheiro que possuem um número de habitantes superior ao introduzido.

```
xp = "//cidade[number(nhabitantes) > " + Integer.valueOf(jTextField14.getText()) +  
"]/nome";
```

Pesquisar por clima

É pedido um clima e a pesquisa mostra todas as cidades existentes no ficheiro que possuem um igual ao introduzido.

```
xp = "//cidade[contains(clima,'" + jTextField14.getText() + "')] /nome";
```

Pesquisar por capitais

É pedido *true* ou *false*, *true* caso se queira procurar as capitais, *false* caso contrário, e a pesquisa mostra todas as cidades existentes no ficheiro que sejam ou não capitais, conforme o valor introduzido.

```
xp = "//cidade[contains(capital,'" + jTextField14.getText() + "')] /nome";
```

Pesquisar por linguagem

É pedido uma linguagem e a pesquisa mostra todas as cidades existentes no ficheiro que utilizam essa linguagem como linguagem oficial.

```
xp = "//linguagens/lingua[contains(.,'" + jTextField14.getText() + "')] /../nome";
```

Pesquisar por cidade geminada

É pedido o nome de uma cidade e a pesquisa mostra todas as cidades existentes no ficheiro que sejam cidades geminadas da cidade introduzida.

```
xp = "//cidadesgeminadas/cidadegeminada[contains(.,'" + jTextField14.getText() +  
"')] /../nome";
```

Ficheiros *XSLT/XQuery*

XSLT

Com uso do XSLT foram implementadas três transformações do documento “*idades.xml*”.

XML >-- HTML fotos das bandeiras dos países

A primeira transformação consiste em criar um novo documento HTML (“*imagensBandeiras.html*”) com as fotos das bandeiras dos países das cidades inseridas no ficheiro, e sem repetições, com recurso ao ficheiro (“*transf1.xsl*”).

Ficheiro Transf1.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="catalogo">
    <html>
      <body>
        <h1>Lista de Bandeiras dos Países</h1>
        <table border="1">
          <tr> <th>Bandeira</th><th>País</th> </tr>
          <xsl:apply-templates>
            <xsl:sort select="pais"/>
          </xsl:apply-templates>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="cidade">
    <tr>
      <td></td>
      <td><xsl:value-of select="pais[not(. = preceding::pais)]"/></td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

XML >-- TXT cidades de um país

A segunda transformação consiste em criar um ficheiro TXT (“*idadesEspanha.txt*”) que mostre a listagem das cidades de um dado país, neste caso foi escolhido Espanha, e com recurso ao ficheiro (“*transf2.xsl*”).

Ficheiro Transf2.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="3.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" />
  <xsl:template match="catalogo">
    <xsl:text>Lista de cidades:&#xa;</xsl:text>
    <xsl:apply-templates select="cidade">
      <xsl:sort select="nome"/>
    </xsl:apply-templates>
  </xsl:template>
  <xsl:template match="cidade">
    <xsl:if test="pais='Espanha'">
      <xsl:text>&#x9;</xsl:text>
      <xsl:value-of select="nome"/>
      <xsl:text>&#xa;</xsl:text>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

XML >-- XML cidades mais populosas

A terceira transformação que consiste na criação de um ficheiro XML ("listagemCidadesPopulosas.xml") com as cinco cidades mais populosas existente no ficheiro, com recurso ao ficheiro ("transf3.xsl").

Ficheiro Transf3.xsl

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="catalogo">
    <cidade>
      <xsl:for-each select="cidade">
        <xsl:sort select="number(nhabitantes)" data-type="number" order="descending"/>
        <xsl:if test="nhabitantes[. > preceding::nhabitantes]">
          <xsl:if test="count(preceding::nhabitantes) < 7">
            <nome>
              <xsl:value-of select="nome"/>
            </nome>
          </xsl:if>
        </xsl:if>
      </xsl:for-each>
    </cidade>
  </xsl:template>
</xsl:stylesheet>
```

Xquery

XML >-- HTML fotos monumentos

Consiste em criar um novo documento HTML ("fotosMonumentos.html") com as fotos dos monumentos da cidade pesquisada pelo utilizador e existente no ficheiro, com recurso ao ficheiro ("monumentos.xql").

Ficheiro monumentos.xql

```
xquery version "1.0";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema";
declare function local:CidadePedida($cidade as xsd:string) as element()*
{
  for $x in doc("cidades.xml")/catalogo/cidade
  where $x/nome = $cidade
  return $x
};
<html><body>
<h1>Imagens de monumentos/landmarks de {doc("pedido.xml")/pedido} </h1>
<table border="1">
<tr><th>Foto</th></tr>
{
  let $v := doc("pedido.xml")/pedido
  let $monumentos := local:CidadePedida($v)/monumentos
  return if (empty($monumentos)) then
    <tr>Pesquisa sem resultados</tr>
  else
    for $x in $monumentos/monumento
    return
      <tr>
        <td></td>
      </tr>
}
</table></body>
</html>
```

XML >-- XML detalhes de cidades de um dado país

Consiste em criar um novo documento XML (“detalhesCidade.xml”) com alguns detalhes das cidades pertencente a um país pesquisado pelo utilizador e existente no ficheiro, com recurso ao ficheiro (“CidadesPais.xql”).

Ficheiro CidadesPais.xql

```
xquery version "1.0";
<queryResult>
{
  for $x in doc("cidades.xml")/catalogo/cidade
  let $v := doc("pedido.xml")/pedido
  where $x/pais[.= $v]
  order by $x/nome
  return <cidade>
      <nome>{$x/nome/text()}</nome>
      <codigopostal>{$x/codigopostal/text()}</codigopostal>
      <website>{$x/website/text()}</website>
</cidade>
}
</queryResult>
```

XML >-- XML cidades com um dado fuso horário

Consiste em criar um novo documento TXT (“cidadesFuso.txt”) com o nome das cidades e o respetivo país com um dado fuso horário pesquisado pelo utilizador e existente no ficheiro, com recurso ao ficheiro (“fuso.xql”).

Ficheiro fuso.xql

```
xquery version "1.0";
<resultados>
{
  for $x in doc("cidades.xml")/catalogo/cidade
  let $v := doc("pedido.xml")/pedido
  where contains($x/fuso,$v)
  order by $x/nome
  return ("&#10;","cidade: ", $x/nome/text(),"----",$x/pais/text())
}
</resultados>
```

XML >-- HTML cidades por linguagem

Consiste em criar um novo documento HTML (“linguagensCidade.html”) com uma lista com os nomes das cidades onde se fala a língua pesquisada pelo utilizador e existente no ficheiro, com recurso ao ficheiro (“linguagens.xql”).

Ficheiro linguagens.xql

```
xquery version "1.0";
declare namespace xsd = "http://www.w3.org/2001/XMLSchema";
declare function local:LinguaPedida($lingua as xsd:string) as element()*
{
  for $x in doc("cidades.xml")/catalogo/cidade
  where $x/linguagens/lingua = $lingua
  return $x
};
<html><body>
<h1>Cidades que falam: {doc("pedido.xml")/pedido}</h1>
<ul>
{
  let $v := doc("pedido.xml")/pedido
  let $cidades := local:LinguaPedida($v)
  return if (empty($cidades)) then
    <tr>Pesquisa sem resultados</tr>
  else
    for $x in $cidades
    return
      <li>{$x/nome/text()}</li>
}
</ul></body>
</html>
```

Conclusão

Com este projeto solidificámos e adquirimos conhecimentos na área de Integração de Dados e fortalecemos competências em Java, XML, DTD, XSD, XSLT e XQuery.