MoviesDB

Universidade de Aveiro

Inês Justo, Miguel Neves,
Rafael Maio, Raquel Rainho

MoviesDB

Departamento de Eletrónica, Telecomunicações e Informática
Engenharia de Dados e Conhecimento

Universidade de Aveiro

(84804) inesjusto@ua.pt
(67453) miguelneves@ua.pt
(84909) rafael.maio@ua.pt
(84891) raquel.a.rainho@ua.pt

10 de novembro de 2019

Conteúdo

Capítulo 1

Introdução

MoviesDB é uma aplicação Web criada com o intuito de apresentar informação relativa ao mundo cinematográfico, tendo sido dividida em quatro partes principais:

- Uma secção de filmes, onde é apresentada uma lista de filmes e respetivos detalhes (tais como o ano de lançamento, géneros, pontuação, entre outros);
- Uma secção de atores, onde são listados todos os atores que participam nos filmes existentes na aplicação;
- Uma secção de realizadores, semelhante à de atores, onde são apresentados todos os realizadores dos filmes existentes;
- Uma secção de notícias, com informação atualizada sobre o mundo cinematográfico.

Para o desenvolvimento da aplicação foram utilizadas as seguintes tecnologias: Python/Django, XML, XML Schema, XPath, XSLT, BaseX, XQuery e RSS.

Capítulo 2

Dados e suas Fontes

2.1 IMDB 5000 Movie Dataset - Movies XML

O dataset "movie_metadata.csv" (no directório "xml") foi obtido na comunidade de data scientists e machine learners, Kaggle. Embora contendo uma grande variedade de informação, este dataset não serviria um dos objetivos propostos, nomeadamente de utilizar dados semi-estruturados, através da linguagem XML. Deste modo, a transformação do CSV em XML era fundamental, pelo que o script "cleancsv.py" e o script "csvtoxml.py" foram elaborados, tendo em conta apenas os filmes que tivessem todos os dados, a seleção dos elementos e atributos, bem como a estrutura do documento, permitindo a criação do ficheiro "movies.xml".
É de notar que, devido à grande quantidade de dados presente no dataset, foi decidido utilizar apenas uma pequena parte deste, tendo sido criado o ficheiro "movies_short.xml". Este foi utilizado para a criação da base de dados do projecto, e assim permitiu maior rapidez de leitura e operações sobre os dados, e consequentemente um carregamento das páginas web mais veloz.

Figura 2.1: Dados de um filme contido no ficheiro "movie_short.xml"

2.2 People XML

Numa fase mais avançada do projeto, tendo em mente a página de informação de cada actor e realizador envolvidos nos filmes apresentados, foi criado o programa "actorsxml.py". Assim, utilizando *Beautiful Soup* - um pacote Python para analisar documentos HTML e XML - e a

biblioteca de python *requests*, foram guardados no ficheiro "people.xml" as fotos e biografias, presentes no imdb, referentes a cada pessoa presente no ficheiro "movies.xml". Estes dois pacotes do Python foram também utilizados em "csvtoxml.py"para ser guardado o url de cada poster correspondente a um filme.

Figura 2.2: Dados de um realizador e de um ator contidos no ficheiro "people.xml"

2.3 RSS

O feed de notícias cinematográficas utilizado encontra-se em `https://www.cinemablend.com/rss/topic/news/movies`. Para a sua utilização dinâmica nesta aplicação, cada vez que o utilizador de MoviesDB pede para aceder às notícias, a aplicação faz um pedido ao `https://www.cinemablend.com/rss/topic/news/movies`, de forma a haver sempre à atualização dos dados.

```
def movies_news_feed(request):
    xml_link = "https://www.cinemablend.com/rss/topic/news/movies"
    xml_file = requests.get(xml_link)
    xslt_name = 'rss.xsl'
    xsl_file = os.path.join(BASE_DIR, 'app/data/xslts/' + xslt_name)
    tree = ET.fromstring(xml_file.content)
    xslt = ET.parse(xsl_file)
    transform = ET.XSLT(xslt)
    newdoc = transform(tree)

    tparams = {
        'content': newdoc,
    }

    return render(request, 'news.html', tparams)
```

Figura 2.3: Atualização do RSS

Capítulo 3

Esquemas dos Dados (XML Schema)

O ficheiro XSD define o esquema de dados para o XML e faz a verificação da validade dos mesmos. Gerou-se automaticamente o XML Schema usando a funcionalidade "Generate XSD Schema from XML File" do JetBrains e alterou-se o mesmo de forma a que estivesse de acordo com o que se pretendia validar no XML.

3.1 Movie XML

Cada filme presente no ficheiro "movies_short.xml" apresenta uma série de atributos. No entanto, tornaram-se obrigatórios apenas alguns por questões de gestão de tempo, apesar de poderem ser todos enquadrados na aplicação web. Foram então escolhidos os que pareciam ser mais relevantes ao contexto da mesma.

```
<xs:attribute type="xs:string" name="color" use="optional"/>
<xs:attribute type="rate" name="rating" use="optional"/>
<xs:attribute type="xs:string" name="country" use="optional"/>
<xs:attribute type="xs:string" name="language" use="optional"/>
<xs:attribute type="xs:string" name="aspect_ratio" use="optional"/>
<xs:attribute type="xs:integer" name="duration" use="required"/>
<xs:attribute type="xs:integer" name="budget" use="optional"/>
<xs:attribute type="xs:integer" name="fb_likes" use="optional"/>
<xs:attribute type="xs:string" name="gross" use="optional"/>
<xs:attribute type="xs:integer" name="num_user_for_reviews" use="optional"/>
<xs:attribute type="xs:integer" name="facenumber_in_poster" use="optional"/>
```

Figura 3.1: Atributos de um filme

Usou-se restrições de valores para o atributo *"rating"* e para o elemento *"genre"* de forma a que estes não possam tomar outros valores que não os definidos nos campos *"enumeration"*.

```
<xs:simpleType name="rate">
    <xs:restriction base="xs:string">
        <xs:enumeration value="PG-13"/>
        <xs:enumeration value="PG"/>
        <xs:enumeration value="G"/>
        <xs:enumeration value="R"/>
        <xs:enumeration value="Approved"/>
        <xs:enumeration value="NC-17"/>
        <xs:enumeration value="X"/>
        <xs:enumeration value="Not Rated"/>
        <xs:enumeration value="Unrated"/>
        <xs:enumeration value="M"/>
        <xs:enumeration value="GP"/>
        <xs:enumeration value="Passed"/>
    </xs:restriction>
</xs:simpleType>
```

Figura 3.2: Atributo rating

```
<xs:complexType name="genresType">
  <xs:sequence>
    <xs:element name="genre" maxOccurs="unbounded" minOccurs="1">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Action"/>
          <xs:enumeration value="Adventure"/>
          <xs:enumeration value="Fantasy"/>
          <xs:enumeration value="Sci-Fi"/>
          <xs:enumeration value="Thriller"/>
          <xs:enumeration value="Romance"/>
          <xs:enumeration value="Animation"/>
          <xs:enumeration value="Comedy"/>
          <xs:enumeration value="Family"/>
          <xs:enumeration value="Musical"/>
          <xs:enumeration value="Mystery"/>
          <xs:enumeration value="Western"/>
          <xs:enumeration value="Drama"/>
          <xs:enumeration value="History"/>
          <xs:enumeration value="Sport"/>
          <xs:enumeration value="Crime"/>
          <xs:enumeration value="Horror"/>
          <xs:enumeration value="War"/>
          <xs:enumeration value="Biography"/>
          <xs:enumeration value="Music"/>
          <xs:enumeration value="Documentary"/>
          <xs:enumeration value="Film-Noir"/>
        </xs:restriction>
```

Figura 3.3: Tipo genreType

Usou-se o elemento <*any*> em conjunto com o *"processContents"* e *"minOccurs"* para acrescentar ao documento XML elementos não especificados no Schema. Desta forma um elemento *"movie"* pode ou não conter elementos entre *"title"* e *"cast"* e mais elementos depois de *"genres"*.

```
<xs:sequence>
  <xs:element type="titleType" name="title"/>
  <xs:any processContents="skip" maxOccurs="unbounded" minOccurs="0"/>
  <xs:element type="castType" name="cast"/>
  <xs:element type="directorType" name="director"/>
  <xs:element type="genresType" name="genres"/>
  <xs:any processContents="skip" maxOccurs="unbounded" minOccurs="0"/>
</xs:sequence>
```

Figura 3.4: Filhos do elemento movie

O tipo simples *"minLength"* foi usado para restringir o valor de alguns elementos, ou seja, não poderão ser preenchidos com uma *string* vazia.

```
<xs:complexType name="titleType">
  <xs:sequence>
    <xs:element name="name" type="minLength"/>
    <xs:element type="xs:integer" name="year"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="minLength">
    <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
    </xs:restriction>
</xs:simpleType>
```

Figura 3.5: Tipo *"minLength"* utilizado no tipo *"titleType"*

O elemento *"cast"* tem o elemento *"main_actors"* que por sua vez faz uso do *"minOccurs"* no elemento *"person"* para que seja obrigatória a existência de três atores principais. Cada elemento *"person"* é composto pelo primeiro e último nome do ator.

```
<xs:complexType name="nameType">
  <xs:sequence>
    <xs:element name="first_name" type="minLength"/>
    <xs:element name="last_name" type="minLength"/>
  </xs:sequence>
</xs:complexType>
```

Figura 3.6: Tipo *"nameType"*

```
<xs:complexType name="personType">
  <xs:sequence>
    <xs:element type="nameType" name="name"/>
    <xs:any processContents="skip" maxOccurs="unbounded" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Figura 3.7: Tipo *"personType"*

```
<xs:complexType name="main_actorsType">
  <xs:sequence>
    <xs:element type="personType" name="person" maxOccurs="unbounded" minOccurs="3"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="castType">
  <xs:sequence>
    <xs:element type="main_actorsType" name="main_actors"/>
  </xs:sequence>
  <xs:attribute type="xs:integer" name="fb_likes" use="optional"/>
</xs:complexType>
```

Figura 3.8: Tipo *"main_ actorsType"*

O elemento director também faz uso do elemento tipo *"personType"* para que este seja composto também pelo primeiro e último nome do diretor.

```
<xs:complexType name="directorType">
  <xs:sequence>
    <xs:element type="personType" name="person"/>
  </xs:sequence>
</xs:complexType>
```

Figura 3.9: Tipo *"directorType"*

3.2 People XML

Para o ficheiro "people.xml" não foi feito um XML Schema pelo simples facto de não haver alteração dos dados do mesmo nesta fase do projeto. Numa continuação do mesmo, seria algo a considerar.

Capítulo 4

Transformações sobre os dados (XSLT)

De forma a facilitar a listagem de filmes, atores e realizadores escolheu-se usar XSLT para transformar os dados no formato XML.

4.1 Listar Actores/Realizadores

A lista dos atores é formada usando o *"template match='/'"* para cada *"person"* em *"cast"* seleciona-se o nome deste (*"name"*), criando também uma referência para a página individual desse ator.

Figura 4.1: XSL actors

Os realizadores são listados de forma idêntica aos atores, mas a iteração é feita por cada realizador.

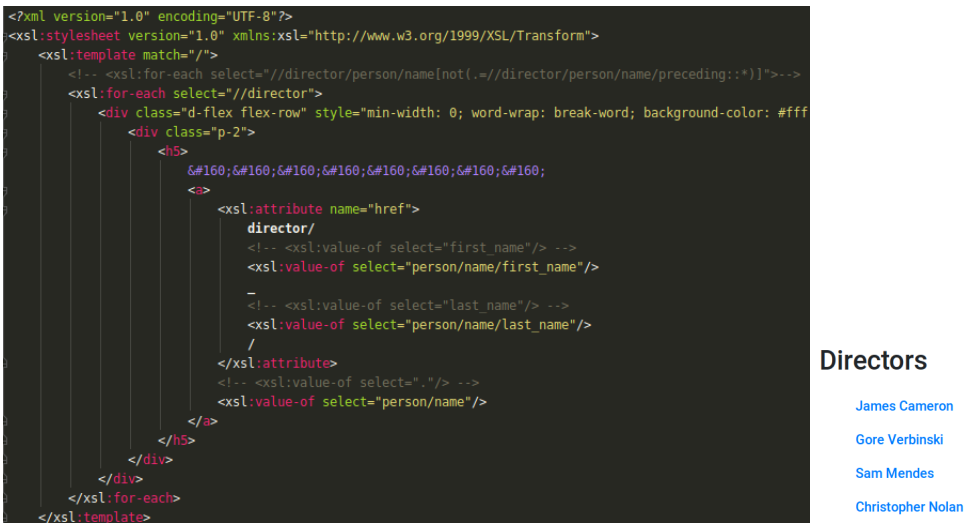

Figura 4.2: XSL directors

4.2 Listar Filmes

A lista dos filmes é feita iterando os filmes, sendo em cada um extraído o valor do título (*"name"*) e ano (*"year"*), elementos filhos do elemento *"title"*, os géneros desse filme (*"genres"*), a sua imagem associada (campo *"poster"*), os nomes dos atores principais, o nome do realizador e a respetiva pontuação (*"score"*).

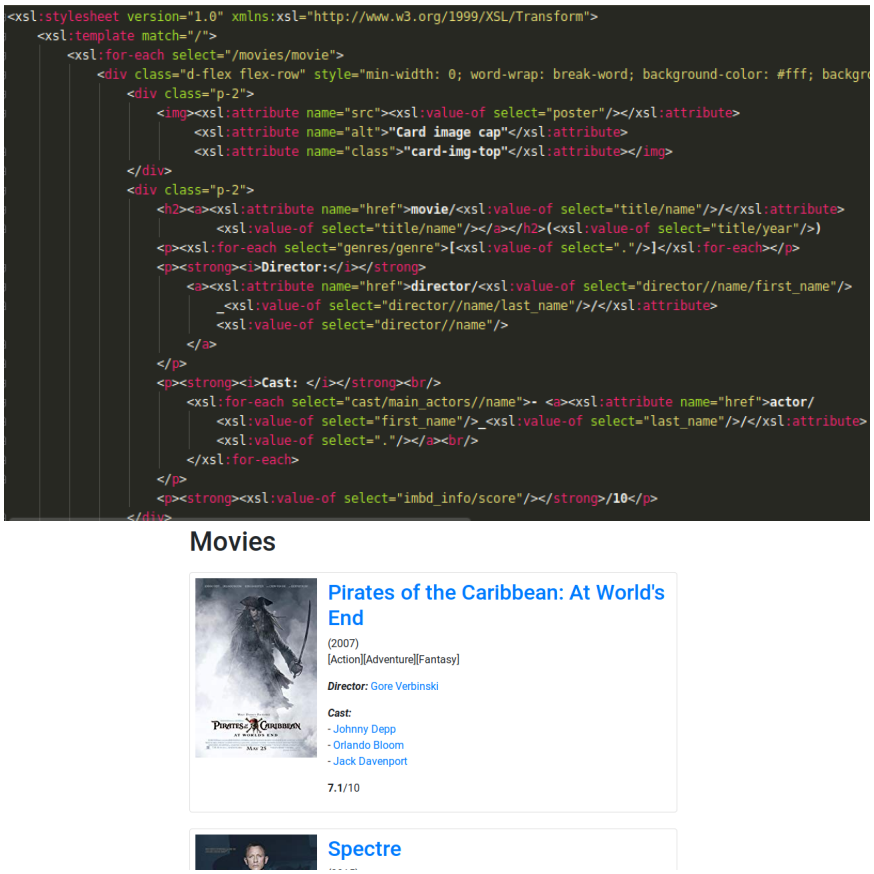

Figura 4.3: XSL movies

Capítulo 5

Operações sobre os Dados

5.1 XQuery

5.1.1 Filtros de pesquisa

À função de pesquisa de filmes é dado como argumento um XML com o elemento-raiz *<query>* e os filhos *<genres>*, *<rating>* e *<year>*. Através deste, são chamadas funções de seleção que retornam todos os filmes que contêm esse elemento (no caso de géneros e ano) ou atributo (no caso de rating) se este não estiver vazio. Neste último caso, é retornada toda a lista de filmes, de modo a que, posteriormente à seleção de todos os dados passados como argumento, sejam selecionados apenas os filmes que tenham como características os resultados das 3 funções de seleção chamadas. Sendo assim, caso sejam selecionados vários géneros, são obtidos filmes com todos ou apenas algum(ns) deles, mas no caso de ser selecionado um rating e/ou ano específicos, os filmes obtidos contêm obrigatoriamente esses valores. (Por exemplo, sendo feita uma filtragem com *Genres: "Action", "Adventure"; Rating: "PG"; Year: "2001"*, são obtidos filmes com a classificação "PG", do ano 2000 e dos géneros de ação e/ou aventura.)

```
declare function movies:apply_filters($query) as element()*{
(: <query>  <genres>  <genre></genre>  </genres>    <rating></rating>    <year></year>  </query> :)
      let $movies := doc("moviesDB")
      let $selected_movies_by_year := movies:selected_year($query)
      let $selected_movies_by_rating := movies:selected_rating($query)
      let $selected_movies_by_genres := movies:selected_genres($query)
      for $movie_y in $selected_movies_by_year//movie
      for $movie_r in $selected_movies_by_rating//movie
      for $movie_g in $selected_movies_by_genres//movie
      where matches(data($movie_y//title/name), data($movie_r//title/name))
            and matches(data($movie_g//title/name), data($movie_r//title/name))
            and matches(data($movie_y//title/name), data($movie_g//title/name))
      return $movie_g
};
```

Figura 5.1: Query para filtro de pesquisa de filmes

```
declare function movies:selected_genres($genres) as element()*{
    <movies>{
      let $movies := doc("moviesDB")
      return if (data($genres//genre[1])="") then
                  for $movie in $movies//movie
                  return $movie
              else
                  for $movie in $movies//movie
                   for $m_genre in $movie//genre
                       for $q_genre in $genres//genre
                       where matches(data($q_genre), data($m_genre))
                       return $movie
    }</movies>
};
```

```
declare function movies:selected_rating($rating) as element()*{
    <movies>{
      let $movies := doc("moviesDB")
      return if  (data($rating//rating)="") then
                  for $movie in $movies//movie
                  return $movie
              else
                  for $movie in $movies//movie
                  where matches(data($movie//@rating), data($rating//rating))
                  return $movie
    }</movies>
};
```

```
declare function movies:selected_year($year) as element()*{
    <movies>{
      let $movies := doc("moviesDB")
      return if (data($year//year)="") then
                  for $movie in $movies//movie
                  return $movie
              else
                  for $movie in $movies//movie
                  where matches(data($movie//year), data($year//year))
                  return $movie
    }</movies>
};
```

Figura 5.2: Queries de seleção

Junto aos filtros de pesquisa, a opção "*Order by*" permite ordenar os resultados por pontuação (*Score*), título (*Title*) ou ano (*Year*). Tal é feito pela função "*selected_filters*", que ordena os resultados da função de filtragem.

```
declare function movies:selected_filters($query, $order) as element()*{
    let $filtered := movies:apply_filters($query)
    return if (data($order)='title') then
      for $name in distinct-values($filtered//title/name)
      let $b := $filtered//title/name[.=$name]
      order by $b[1]/../..//title/name
      return $b[1]/../..
    else if (data($order)='score') then
      for $name in distinct-values($filtered//title/name)
      let $b := $filtered//title/name[.=$name]
      order by $b[1]/../..//imbd_info/score descending
      return $b[1]/../..
    else if (data($order)='year') then
      for $name in distinct-values($filtered//title/name)
      let $b := $filtered//title/name[.=$name]
      order by $b[1]/../..//title/year descending
      return $b[1]/../..
    else for $name in distinct-values($filtered//title/name)
      let $b := $filtered//title/name[.=$name]
      return $b[1]/../..
};
```

Figura 5.3: Query para a ordenação dos resultados da filtragem

5.1.2 Pesquisa por texto

O utilizador pode pesquisar na página inicial, a responsável por listar os filmes, por palavras contidas nos títulos ou nas palavras-chave (*plot keywords*) de um determinado filme. É usada a função "dist_searcher", responsável por retornar os filmes distintos que por sua vez usa a função "searcher", já esta responsável por retornar os filmes enquadrados na pesquisa do utilizador.

```
declare function movies:dist_searcher($search) as element()*{
  let $s := movies:searcher($search)
  for $dist_names in distinct-values($s//title/name)
  let $d := $s//title/name[.=$dist_names]
  return $d[1]/../..
};
```

Figura 5.4: XQuery pesquisa de filmes com filmes iguais removidos

```
declare function movies:searcher($search) as element()*{
  for $bs in collection('moviesDB')/movies/movie
  for $p in $bs/plot_keywords/keyword
  where contains(lower-case($bs/title/name),lower-case($search)) or contains(lower-case($p),lower-case($search))
  return $bs
};
```

Figura 5.5: XQuery pesquisa de filmes

O utilizador pode ainda pesquisar pelo nome de um ator ou de um realizador específico, sendo estas pesquisas bastante idênticas. Ambas as pesquisas são feitas inserindo o nome, ou parte deste, de um determinado ator, ou realizador. A primera usa a função *"dist_ searchActor"* (responsável por retornar apenas os atores distintos) que tem a função auxiliar *"searchActor"* que vai verificar em todas as pessoas (*person*) que têm a profissão *Actor* e o nome correspondente com a pesquisa feita pelo utilizador.
A pesquisa dos realizadores por sua vez procura os que têm a profissão *Movie Director*.
Cada uma das pesquisas é realizada na secção correspondente, "*Actors*"e "*Directors*".

```
declare function movies:searchActor($search) as element()*{
  let $people := doc("moviesDB")//person
  for $person in $people
  where matches(data($person//profession), "Actor") and contains(lower-case($person//name),lower-case($search))
  return $person//name
};
```

Figura 5.6: XQuery pesquisa de atores

```
declare function movies:dist_searchActor($search) as element()*{
  let $s := movies:searchActor($search)
  for $dist_names in distinct-values($s)
  let $d := $s[.=$dist_names]
  return $d[1]/..
};
```

Figura 5.7: XQuery pesquisa de atores com atores iguais removidos

```
declare function movies:searchDirector($search) as element()*{
   let $people := doc("moviesDB")//person
  for $person in $people
  where matches(data($person//profession), "Movie Director") and contains(lower-case($person//name),lower-case($search))
  return $person//name
};
```

Figura 5.8: XQuery pesquisa de realizadores

```
declare function movies:dist_searchDirector($search) as element()*{
  let $s := movies:searchDirector($search)
  for $dist_names in distinct-values($s)
  let $d := $s[.=$dist_names]
  return $d[1]/../..
};
```

Figura 5.9: XQuery pesquisa de realizadores com realizadores iguais removidos

5.1.3 Apresentação dos dados

Cada filme na base de dados tem uma série de elementos e atributos, onde se escolheu os que se consideraram mais relevantes para serem apresentados. Para isto foram usadas várias queries, cada uma responsável por retornar uma determinada informação do filme correspondente.

```
declare function movies:get_movie_year($movie_name) as item(){
  let $movie := doc("moviesDB")//movie[title/name=$movie_name]
  return $movie//year
};
```

Figura 5.10: XQuery retornar ano de um filme

```
declare function movies:get_movie_main_actors($movie_name as xs:string) as element()*{
    let $movie := doc("moviesDB")//movie[title/name=$movie_name]
    for $actor in $movie//main_actors//name
      return <actor>{string-join(($actor/first_name/text(), $actor/last_name/text())," ")}</actor>
};
```

Figura 5.11: XQuery retornar atores de um filme

Tal como são apresentados os dados de um filme, é feito o mesmo para os atores e realizadores, tendo cada um uma página com as suas informações (podendo esta estar quase sem informações caso o ator/realizador não exista na base de dados *peopleDB*). Para isto são também usadas algumas queries para retornar estas informações.

```
declare function movies:get_movies_by_actor($a_first_name, $a_last_name) as element()*{
  for $movie in  doc("moviesDB")//movie
    for $actors in $movie//main_actors//person
      where matches(data($actors//first_name), $a_first_name) and matches(data($actors//last_name), $a_last_name)
      return $movie
};
```

Figura 5.12: XQuery retornar filmes de um determinado ator

```
declare function people:get_bio($actor) as element()*{
  for $name in doc("peopleDB")//name
  where contains(data($name), data($actor//first_name)) and contains(data($name), data($actor//last_name))
  return $name/../bio
};
```

Figura 5.13: XQuery retornar a biografia de um ator/realizador

5.2 XQuery Update

5.2.1 Adição de um filme

Usando o botão *"New movie"* o utilizador é encaminhado para uma nova página onde lhe é pedido para inserir algumas informações sobre o novo filme, sendo obrigatório o preenchimento de alguns dos campos.
Antes da inserção do filme na base de dados é verificado em conjunto com o XML Schema, referido no Capítulo 3, a validade dos dados inseridos, de forma a que o utilizador não possa inserir dados com erros, como por exemplo, um género que não exista ou inserir letras no campo do ano.

Exemplo de dados válidos para um novo filme a inserir:

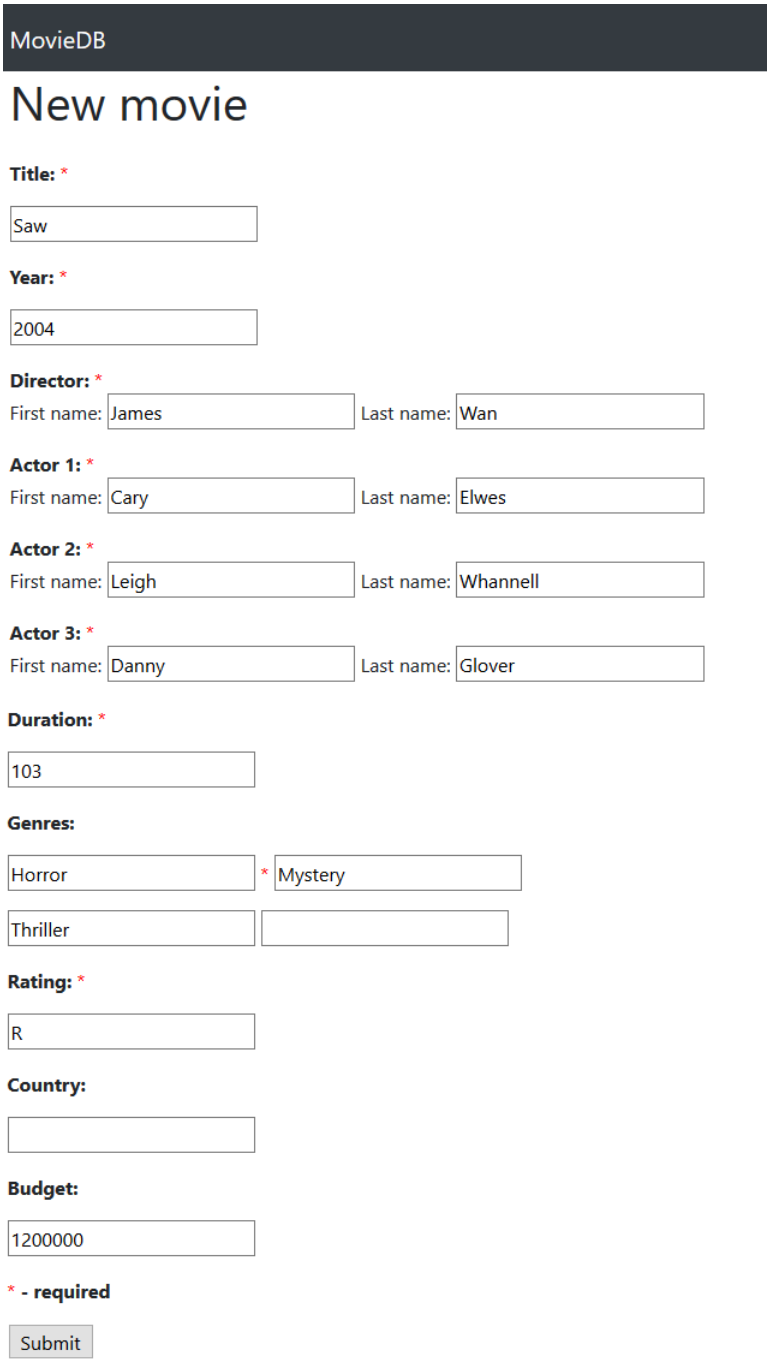

Figura 5.14: Página para inserção de um filme na base de dados

Se os dados não forem inseridos correctamente aparecerá uma mensagem de erro quando o utilizador clica no botão *"Submit"*. Caso contrário esse novo filme vai ser inserido na base de dados usando XQuery Update, em que as informações são passadas do formato de um dicionário para XML, que é passado como parâmetro para a função *"ins_ movie"*.

```
declare updating function movies:ins_movie($movie){
  let $bs := collection('moviesDB')//movies
  let $node := $movie//movie
  return insert node $node as first into $bs
};
```

Figura 5.15: Query para inserir um filme na base de dados

5.2.2 Remoção de um filme

Na página individual de cada filme existe o botão *"Delete"*, que após um *prompt* de confirmação apaga esse mesmo filme da base de dados. Para isso dá-se mais uma vez uso ao XQuery Update, em que o parâmetro passado para a função *"del_ movie"* é uma *string* com o nome do respetivo filme.

```
declare updating function movies:del_movie($movie){
  let $movie := doc('moviesDB')//movie[title/name=$movie]
  return delete node $movie
};
```

Figura 5.16: Query para remover um filme na base de dados

Capítulo 6

Funcionalidades da Aplicação (UI)

Tal como referido no Capítulo 1, as funcionalidades da aplicação podem agrupar-se em quatro áreas principais, focando-se em filmes, atores, realizadores e notícias. É de referir que a navegabilidade entre estas pode ser realizada de diversas formas, pois todos os títulos de filmes e nomes de atores e realizadores redirecionam o utilizador para a sua página detalhada.

6.1 Filmes (*Movies*)

- Visualizar a lista de filmes existentes na base de dados;

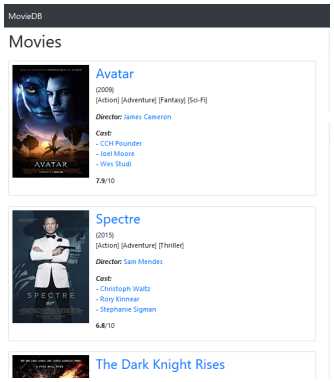

Figura 6.1: Exemplo de vista de filmes

- Pesquisar por filmes de um determinado género (*genre*), classificação (*rating*), ano de lançamento (*year*) ou qualquer combinação destes;
- Ordenar os filmes por título (*title*), ano de lançamento (*year*) ou pontuação (*score*);

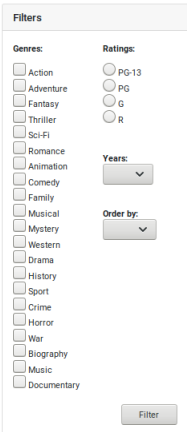

Figura 6.2: Filtros e ordenação

- Pesquisar por texto, contido no título ou palavras-chave (*plot keywords*) de um filme;

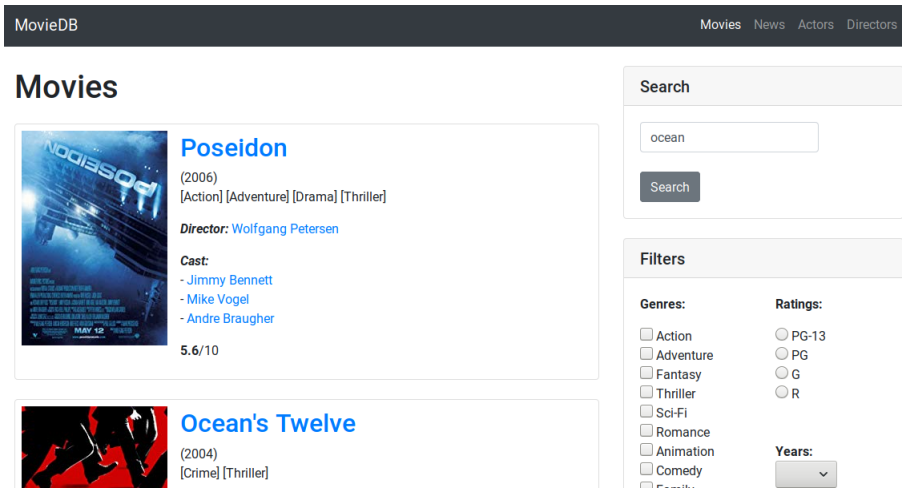

Figura 6.3: Pesquisa de filmes com o termo "ocean"

- Adicionar um filme à base de dados, utilizando o botão respetivo (*New Movie*);

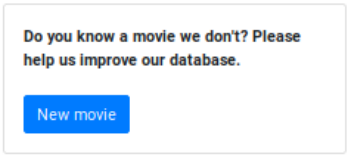

Figura 6.4: Botão para adicionar um novo filme

- Aceder aos detalhes de um filme específico, clicando no seu título;
- Remover um filme da base de dados, utilizando o botão presente nos detalhes do filme (*Delete*).

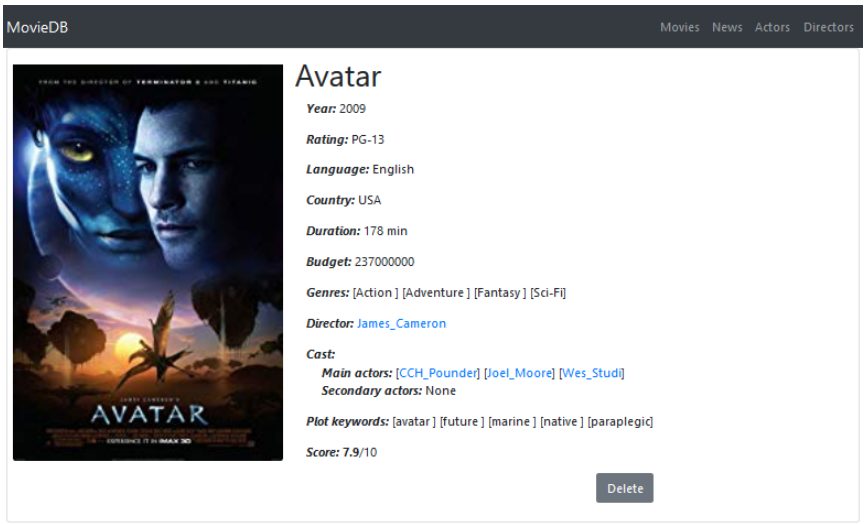

Figura 6.5: Página detalhada do filme "Avatar"

6.2 Atores (*Actors*)

- Visualizar todos os atores dos filmes existentes na base de dados;

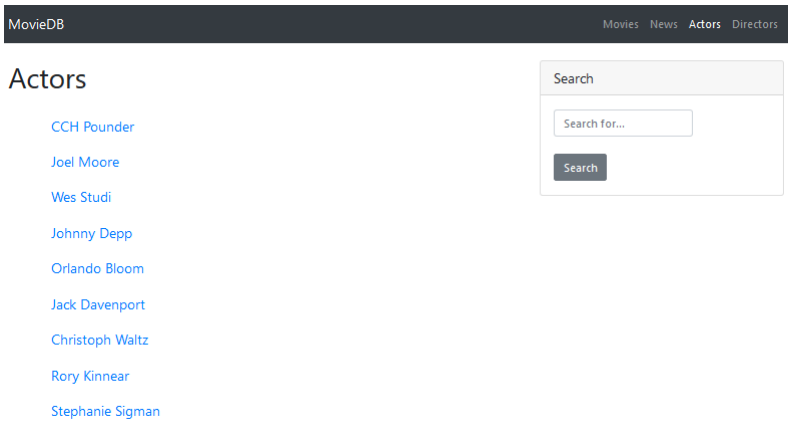

Figura 6.6: Exemplo de atores que participam em filmes existentes na base de dados

- Pesquisar atores pelo seu nome;

Figura 6.7: Pesquisa de atores cujo nome contém "Joel"

- Aceder ao perfil de um ator (que contém a sua mini-biografia e uma fotografia), clicando no seu nome;
- Verificar, no perfil do ator, quais os filmes em que participa.

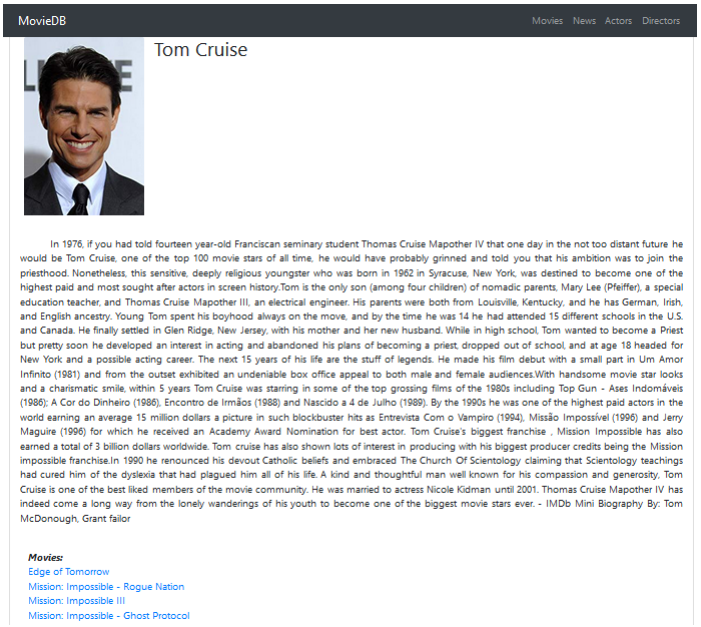

Figura 6.8: Detalhes do ator "Tom Cruise" e lista de filmes em que participou

6.3 Realizadores (*Directors*)

- Visualizar todos os realizadores dos filmes existentes na base de dados;

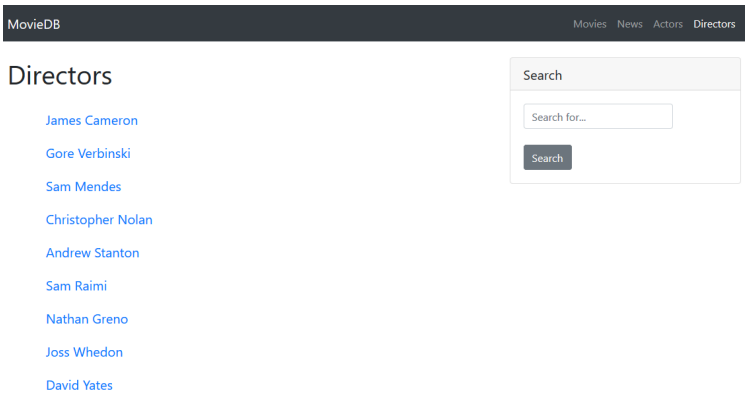

Figura 6.9: Exemplo de realizadores que dirigem filmes existentes na base de dados

- Pesquisar realizadores pelo seu nome;

Figura 6.10: Pesquisa de realizadores cujo nome contém "cameron"

- Aceder ao perfil de um realizador (que contém a sua mini-biografia e uma fotografia), clicando no seu nome;
- Verificar, no perfil do realizador, quais os filmes que dirige.

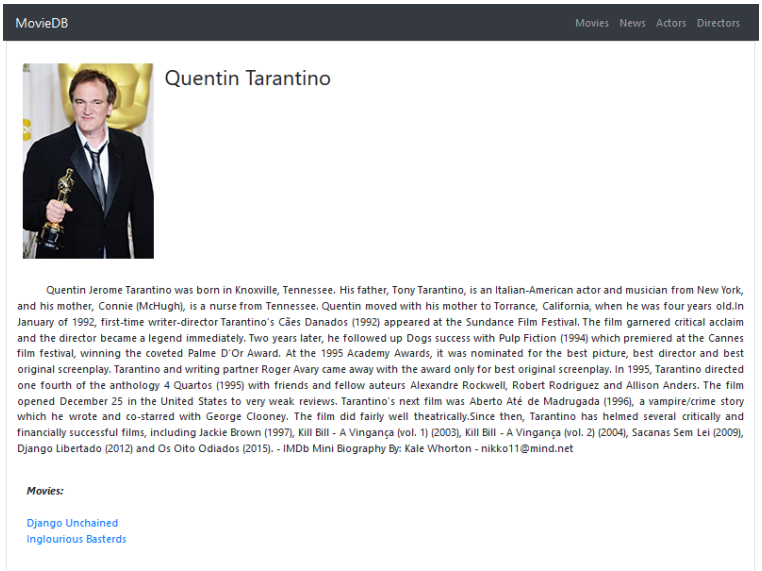

Figura 6.11: Detalhes do realizador "Quentin Tarantino" e lista de filmes que dirigiu

6.4 Notícias (*News*)

- Apresentar cabeçalhos das últimas notícias cinematográficas, com redirecionamento para a fonte destas, onde é possível consultar a notícia completa.

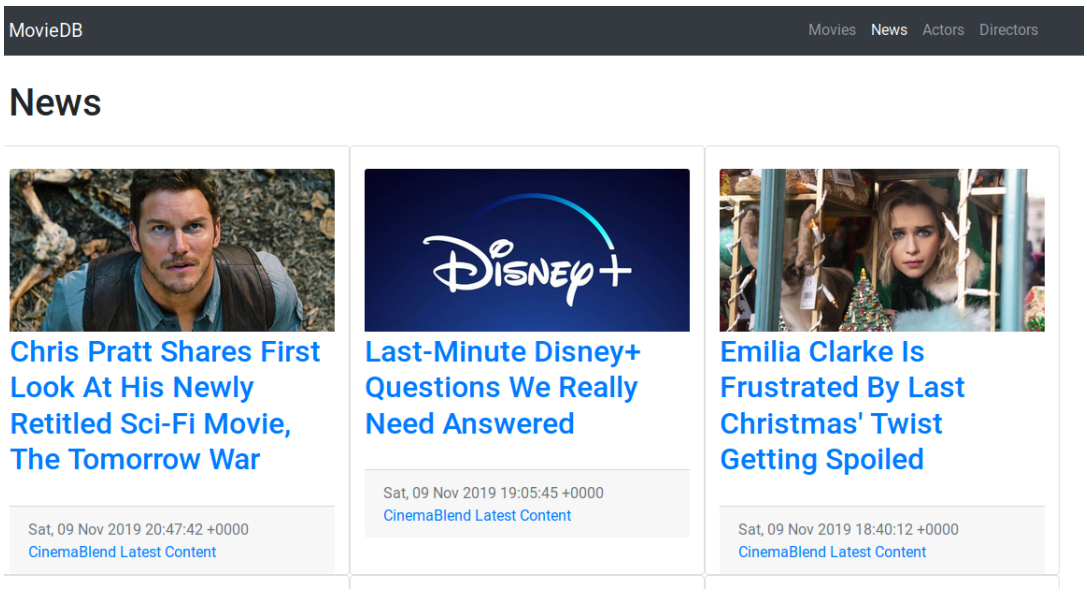

Figura 6.12: Exemplo de cabeçalhos de notícias cinematográficas

Capítulo 7

Conclusões

O projecto apresentado permitiu atingir todos os objectivos propostos, pois foi dado um propósito real para todas as tecnologias propostas e utilizadas.
Foi desenvolvido em grande parte o conhecimento na área da informação e da organização e representação de dados, um ponto fundamental neste e em qualquer projeto na área das tecnologias de informação.
Em suma, a aplicação desenvolvida permite um fácil e intuitivo acesso às principais informações sobre filmes, atores e realizadores, através de boa organização, apresentação de dados e navegabilidade.

Capítulo 8

Configuração para Executar a Aplicação

1. Criar as bases de dados necessárias ao projecto, segundo um dos seguintes métodos:
   - Através da aplicação BaseX, criar uma base de dados com o nome "moviesDB" utilizando o ficheiro "movies_short.xml" e uma outra com o nome "peopleDB" utilizando o ficheiro "people.xml" (ambos se encontram em "webproj\app\data");
     OU
   - Colocar o conteúdo do ficheiro fornecido "baseX_data.zip" no directório onde se encontra instalada a aplicação BaseX (a pasta "data" contém as duas bases de dados já criadas).
2. Instalar as dependências do projecto listadas com a respectiva versão no ficheiro requirements.txt (no directório "webproj") - através do PyCharm ou utilizando o comando *"pip install -r requirements.txt"*.
3. Inicializar o servidor de BaseX, seguido da aplicação em si, através do PyCharm, e aceder a *http://127.0.0.1:8000/* no browser.

Acrónimos

XML Extensible Markup Language

CSV Comma-Separated Values

RSS Really Simple Syndication

XSLT Extensible Stylesheet Language Transformations

XSD XML Schema Definition

Bibliografia

[1] Kaggle (https://www.kaggle.com/carolzhangdc/imdb-5000-movie-dataset)

[2] Cinema Blend (https://www.cinemablend.com/)