



ISEL

INSTITUTO SUPERIOR DE
ENGENHARIA DE LISBOA

Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e
Multimédia (LEIM)
Tecnologias de Informação (TI)
Semestre 1819SI



VIDTUBE

Trabalho prático nº 3 – XPath e XSLT

Docente Eng.º António Teófilo

David Nunes	43033
Luís Fonseca	45125
João Santos	45156

Lisboa, 27 de Novembro de 2018

Índice de matérias

1.	Introdução / Objetivos.....	1
2.	Expressões XPath.....	2
3.	Transformações XSLT	8
i.	XML para XHTML.....	8
ii.	XML para XML.....	10
4.	<i>Framework</i> CSS	12
5.	Vantagens e Desvantagens da solução apresentada.....	12
6.	Conclusões	12

Índice de exemplos

Exemplo 1 - À esquerda as alterações feitas no ficheiro DTD e à direita um exemplo da mudança no XML.....	4
Exemplo 2 - À esquerda as alterações feitas no DTD e à direita um exemplo do resultado das alterações no XML.....	7

Índice de figuras

Figura 1 - Exemplos XPath para o número de vídeos da plataforma	2
Figura 2 - À esquerda a lista de vídeos publicada pelo utilizador 2, à direita o utilizador 4 sem listas publicadas.....	3
Figura 3 - À esquerda o número de listas a que o utilizador 2 está subscrito, à direita as do utilizador 4	3
Figura 4 - Acima o título do último vídeo publicado pelo utilizador 1 recorrendo aos eixos de XPath. À esquerda o título do último vídeo publicado pelo utilizador 2 recorrendo a outra forma de navegação.....	4
Figura 5 - Acima o número total de gostos que os vídeos publicados pelo utilizador 2 obtiveram. À esquerda os do utilizador 1. Novamente verifica-se a possibilidade de usar expressões diferentes para o mesmo fim.....	5
Figura 6 - À esquerda observamos o título dos vídeos em que o utilizador 1 colocou "gosto", à direita o do utilizador 3.	5
Figura 7 – Acima podemos observar o resultado da expressão desenvolvida. No primeiro caso, como o utilizador 2 só tem 2 vídeos publicados o resultado é “false”, já no segundo caso, em que alterámos para mais do que 1 vídeo publicado, aparece o nome do utilizador.....	6
Figura 8 - À esquerda a expressão que permite obter todos os comentários efetuados no mês de fevereiro. Abaixo esta expressão alterada de forma a mostrar um resultado compatível com os comentários existentes	7
Figura 9 - Acima os nomes dos utilizadores que fizeram comentários num dado vídeo, no primeiro caso o vídeo em questão tinha um comentário, no segundo caso o vídeo não tinha comentários.....	8
Figura 10 - À esquerda o código XSLT usado para gerar a página XHTML com informação média dos utilizadores. À direita a informação detalhada.	9
Figura 11 - Acima observa-se o código XSLT usado para gerar a página com informação média acerca dos vídeos presentes no website. À esquerda o código XSLT para a informação detalhada de cada vídeo	9
Figura 12 - À esquerda observamos o código XSLT com a informação média à cerca das playlists. Abaixo temos o código XSLT com informação detalhada relativa a cada uma das playlists	10
Figura 15 – Possível resolução de um documento XSLT para gerar um documento XML.....	11
Figura 14 - XML gerado através do documento XSLT criado	11
Figura 13 - Comprovação da validação do documento XML.....	11
Figura 16 - Documento DTD usado para a validação do documento XML.....	12

1. Introdução / Objetivos

Para o último trabalho proposto o principal objetivo pretendido era de, com base nos desenvolvimentos obtidos nos trabalhos anteriores, implementar os vários grupos de trabalho propostos, entre eles a utilização de expressões XPath para obter informações com base no ficheiro XML da plataforma, utilização de transformações XSLT para gerar documentos XHTML e XML a partir dos documentos previamente desenvolvidos e por último a utilização de um *framework* CSS. As expressões XPath desenvolvidas deveriam obter as informações pedidas no enunciado, tendo de para isso ser válidas e corretas, de forma a obter os *outputs* pretendidos. Nas transformações XSLT era pretendido, mediante certas alíneas apresentadas, criar uma transformação XSLT que gerasse um documento XHTML, baseado nos resultados do XPath, contendo a informação pretendida e utilizando estilos suportados em ficheiros externos CSS com o objetivo final de visualizar os resultados obtidos num *browser*. De seguida pretendia-se desta vez uma transformação XSLT que a partir do conteúdo do XML inicial gerasse um novo documento XML com a lista dos utilizadores e os vídeos publicados pelos mesmos. Para a resolução de ambos os passos foram ainda apresentadas diretrizes a cumprir. Por último foi pedida a utilização de um *framework* CSS, que para tal foi sugerido criar uma diretoria contendo todos os ficheiros apresentando uma solução equivalente à obtida nas transformações XSLT mas desta vez fazendo uso de um *framework* CSS em todo o site.

Para este relatório iremos desenvolver as soluções pretendidas apresentando conteúdo explicativo do raciocínio desenvolvido de forma a obter os resultados pretendidos, para tal iremos recorrer a resumos explicativos, figuras e exemplos à medida que estes sejam oportunos.

2. Expressões XPath

No enunciado foi pedido primeiramente que, com base no ficheiro XML da plataforma desenvolvido ao longo dos trabalhos previamente realizados, o grupo desenvolvesse expressões XPath que permitissem obter as informações pedidas. Desta feita, apresentamos de seguida cada uma das informações pedidas seguidas da expressão desenvolvida que permite obter essa informação.

i. Número total de vídeos publicados

Para obter o número total de vídeos publicados na plataforma é necessário procurar por toda a plataforma pelos vídeos publicados, para tal utilizámos “//video/@Titulo” para que a expressão considerasse nós a qualquer profundidade, identificando os nós contendo o atributo “Titulo” que fizessem parte do elemento “video”. Para efetuar a contagem dos resultados obtidos utilizou-se a função “count(...)”, sendo então a expressão utilizada para resolver o problema “count(//video/@Titulo)”. Poderia, no entanto, ser utilizada uma expressão mais longa, tal como “count(//Tivid/videos/video/@Titulo)”, que apesar de se obter o mesmo resultado, as expressões “Tivid” e “videos” são redundantes, pois ao utilizar a origem “//” o XPath procura pelos vídeos da plataforma por todos os nós, a qualquer profundidade a que eles se encontrem, pelo que basta usar a expressão mais curta.

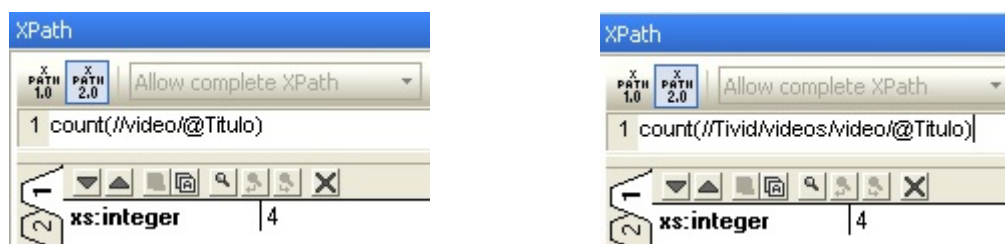


Figura 1 - Exemplos XPath para o número de vídeos da plataforma

ii. Listas de vídeos publicadas por um utilizador

Para obter as listas de vídeos publicadas por um utilizador específico iniciamos a procura utilizando a expressão “//criador” pois este é um elemento exclusivo às *playlists* que identifica o utilizador que criou (publicou) a *playlist*. De seguida é necessário criar um predicado para filtrar os elementos obtidos, de forma a só obter no resultado os elementos relativos ao utilizador pretendido, desta forma adicionamos “[@id=...]” à expressão anterior, identificando o utilizador pretendido. Com esta expressão já temos o utilizador pretendido isolado, de seguida só precisamos que a expressão nos apresente o nome da *playlist* que o utilizador selecionado criou. Para isso recorreremos à expressão “..” para selecionar o nó

pai do nó corrente seguido então do atributo “TT” (atributo que identifica o título da *playlist*) através da expressão “@TT”. Obtemos então a expressão geral “//criador[@id=“...”]/../@TT” que, caso queiramos saber quais as listas de vídeo publicadas pelo utilizador 1, substituímos na expressão “//criador[@id=“U01”]/../@TT” obtendo então o nome da *playlist* publicada “Pra ver”. Caso o utilizador não tenha publicado nenhuma lista de vídeos o resultado obtido é vazio (“No results”).

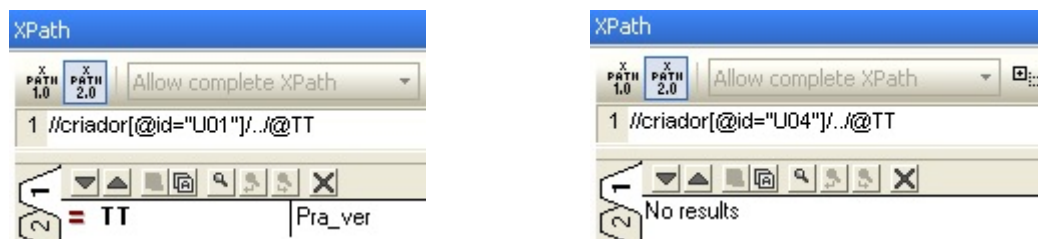


Figura 2 - À esquerda a lista de vídeos publicada pelo utilizador 2, à direita o utilizador 4 sem listas publicadas

iii. Número de listas subscritas por um utilizador

Para obter o número total de listas a que um dado utilizador se encontra subscrito é necessário isolar o nó que contém a informação relativa ao subscritor da *playlist*. Como no XML criado pelo grupo foram utilizados *namespaces* para os vários estilos de subscrição à lista de vídeos (administrador, editor e subscritor), era necessário isolar o nó que continha só a informação do subscritor (sem privilégios de administrador ou editor), pelo que, para tal, recorreu-se ao uso da expressão “//subscritor[@id= “...” and @ultimo_vid]”, pois apenas o nó dos subscritores contém esses dois atributos. O resto da expressão foi facilmente obtida, tendo sido simplesmente adicionada a função “count(...)” de forma a obter um número que diga a quantas *playlists* o utilizador selecionado se encontra somente subscrito. A expressão final utilizada foi então “count(//subscritor[@id=“...” and @ultimo_vid])”.



Figura 3 - À esquerda o número de listas a que o utilizador 2 está subscrito, à direita as do utilizador 4

iv. Título do último vídeo publicado por um utilizador

Para obter o último vídeo publicado por um dado utilizador o recorreu-se ao uso da função de posicionamento “last()”, de forma a obter o último nó correspondente ao último vídeo publicado. Desta forma foi criada uma expressão para obter o título de todos os vídeos publicados por um dado utilizador (neste caso foi decidido usar o eixo XPath “parent”), “//referenciaUtilizador[@id=“...”]/parent::video/@Titulo”, e de seguida foi então adicionada a função “last ()” de forma a obter a posição do último nó, obtendo-se desta forma então a expressão final “(//referenciaUtilizador[@id=“...”]/parent::video/@Titulo)[last()]”.

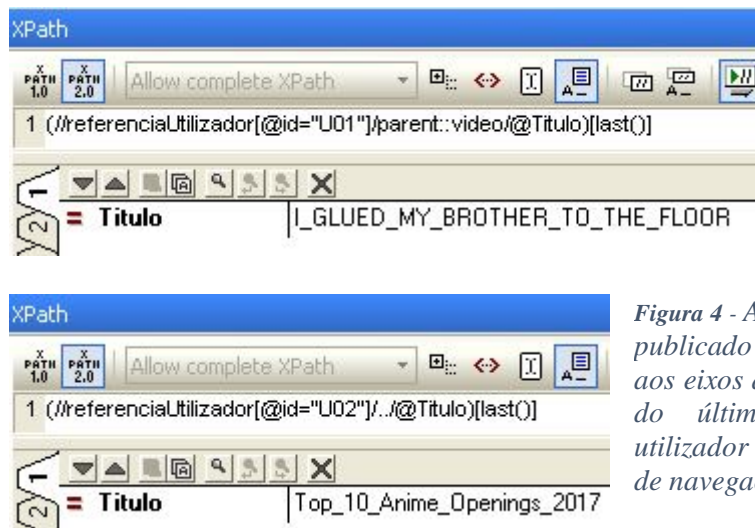


Figura 4 - Acima o título do último vídeo publicado pelo utilizador 1 recorrendo aos eixos de XPath. À esquerda o título do último vídeo publicado pelo utilizador 2 recorrendo a outra forma de navegação

v. Número total de gostos que têm todos os vídeos de um utilizador

Para obter o número total de gostos que todos os vídeos que um dado utilizador publicou obtiveram foi necessário alterar o XML e o DTD de forma a que no seu conteúdo existisse essa informação, que, até ao trabalho anterior, não tinha sido necessária. Para tal, foi criado o elemento “reacao_video” contendo os atributos “id” e “gosto” ou “nao_gosto”. No DTD foram adicionadas as linhas de código necessárias para que ambos os ficheiros (XML e DTD) continuassem bem formados e válidos.

<pre>(...)</pre> <pre><!ELEMENT reacao_video EMPTY></pre> <pre><!-- ATTLIST reacao_video</pre> <pre>id IDREF #REQUIRED</pre> <pre>gosto CDATA #IMPLIED</pre> <pre>nao_gosto CDATA #IMPLIED></pre> <pre>(...)</pre>	<pre>(...)</pre> <pre><referenciaVideo Titulo="Top_10_Anime_Openings_2017"/></pre> <pre><reacao_video id="U01" gosto="yes"/></pre> <pre><reacao_video id="U03" nao_gosto="yes"/></pre> <pre><reacao_video id="U04" gosto="yes"/></pre> <pre><dataInserção>2018/09/18</dataInserção></pre> <pre>(...)</pre>
--	--

Exemplo 1 - À esquerda as alterações feitas no ficheiro DTD e à direita um exemplo da mudança no XML

De seguida procedeu-se então à criação da expressão que nos permitiria obter os resultados pretendidos, e, para tal, começámos por isolar os nós correspondentes aos vídeos que um dado utilizador publicou através da expressão “//referenciaUtilizador[@id=“...”]”. Tendo obtido os nós pretendidos, só ficava a faltar obter os gostos que cada vídeo obteve, e para tal recorreu-se a uma expressão que voltasse ao nó pai para de seguida obter o nó correspondente aos gostos do vídeo (seria também possível recorrer à hierarquia dos eixos XPath, utilizando a expressão “preceding-sibling”). Por último efetuava-se a contagem dos nós obtidos com a função “count(...)” obtendo-se assim a expressão final “count(//referenciaUtilizador[@id=“..."]/..reacao_video[@gosto])”.

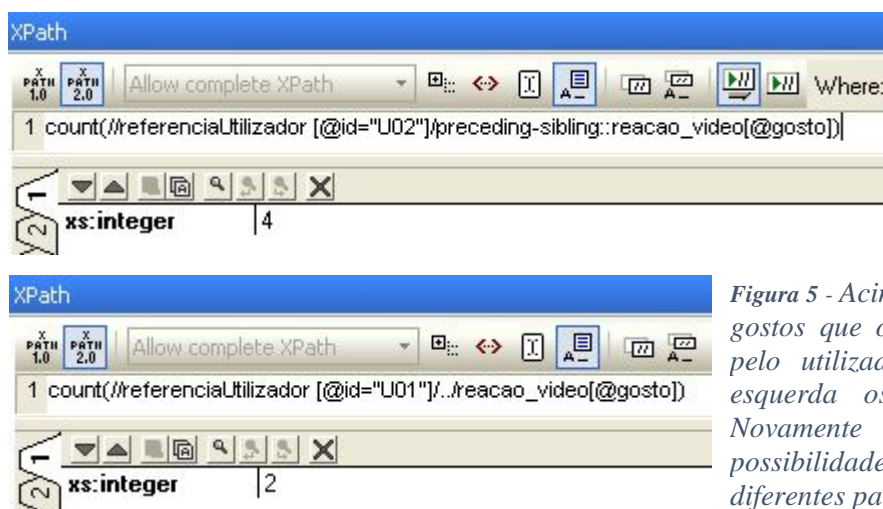


Figura 5 - Acima o número total de gostos que os vídeos publicados pelo utilizador 2 obtiveram. À esquerda os do utilizador 1. Novamente verifica-se a possibilidade de usar expressões diferentes para o mesmo fim.

vi. Títulos dos vídeos que um utilizador indicou “Gosto”

Para obter os títulos dos vídeos em que um dado utilizador colocou gosto bastava isolar os nós correspondentes à reação do vídeo correspondente ao gosto e correspondente ao utilizador selecionado. Para tal foi criada a expressão “//reacao_video[@gosto and @id=“...”]” que isolava os nós correspondentes aos vídeos em que um dado utilizador fez gosto, e, de seguida, só necessitávamos de obter o título desses mesmos vídeos, pelo que navegámos até ao nó pai e obtivemos o atributo “Titulo” correspondente ao vídeo. Obtemos então a expressão final “//reacao_video[@gosto and @id=“..."]/..@Titulo”.



Figura 6 - À esquerda observamos o título dos vídeos em que o utilizador 1 colocou "gosto", à direita o do utilizador 3.

vii. Nomes dos utilizadores com mais de 5 vídeos publicados

Para obter os nomes dos utilizadores com mais de 5 vídeos publicados recorreu-se a duas expressões distintas relacionadas através da função com controlo de fluxo “if”, tendo então criado uma expressão que nos permitisse verificar se um dado utilizador tinha publicado mais do que 5 vídeos (obtendo um *output* “true” ou “false”) e de seguida obter o nome desse utilizador, caso este tivesse realmente publicado mais de 5 vídeos. Para tal foi então desenvolvida a expressão “count(//referenciaUtilizador[@id=“...“])>5” para verificar os vídeos publicados e de seguida a expressão “//utilizador[@id=“...“]/nome” para nos dar o nome do utilizador. A expressão final desenvolvida pelo grupo foi então “if (count(//referenciaUtilizador[@id=“...“])>5) then //utilizador[@id=“...“]/nome else “false”.

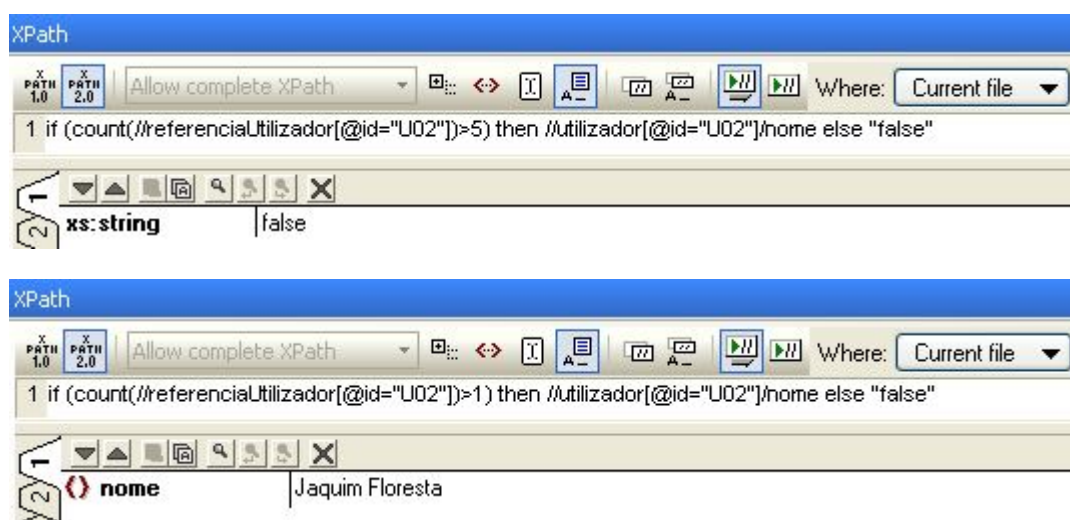


Figura 7 – Acima podemos observar o resultado da expressão desenvolvida. No primeiro caso, como o utilizador 2 só tem 2 vídeos publicados o resultado é “false”, já no segundo caso, em que alterámos para mais do que 1 vídeo publicado, aparece o nome do utilizador.

viii. Listar os comentários do mês de Fevereiro deste ano do(s) vídeo(s) com um dado título

Para obter os comentários do mês de fevereiro de 2018 relativos a um dado vídeo era somente necessário obter os comentários feitos no mês de fevereiro de 2018, que no caso do nosso XML não existe nenhum comentário correspondente (pelo que foi então usado um mês que correspondesse a um comentário). Para conseguir obter o resultado pretendido foi no entanto necessário fazer algumas alterações no XML e no DTD, pois apesar de inicialmente ter sido pensado usar a função “dateTime()” esta estava constantemente a apresentar erros que o grupo infelizmente não conseguiu resolver, tendo então recorrido a uma opção mais simples e alterando a forma como se apresentava a data dos comentários. Para tal foi então criado o elemento vazio “datacomentario” contendo os atributos “dia”, “mes” e “ano”.

<pre>(...)</pre> <pre><!ELEMENT datacomentario EMPTY></pre> <pre><!ATTLIST datacomentario</pre> <pre> dia CDATA #REQUIRED</pre> <pre> mes CDATA #REQUIRED</pre> <pre> ano CDATA #REQUIRED></pre> <pre>(...)</pre>	<pre>(...)</pre> <pre><Comentario></pre> <pre><comentário idcom="_01" id="U02"</pre> <pre>Titulo="Top_10_Anime_Openings_2017"/></pre> <pre><datacomentario dia="21" mes="06" ano="2018"/></pre> <pre><reacao_comentario></pre> <pre>(...)</pre>
--	---

Exemplo 2 - À esquerda as alterações feitas no DTD e à direita um exemplo do resultado das alterações no XML

Através dessa alteração foi então possível criar a expressão “//datacomentario[@ano="2018" and @mes="02"]/following-sibling::texto” que iria obter os nós correspondentes a todos os comentários efetuados no mês de fevereiro. Como no nosso XML não havia nenhum resultado correspondente ao mês de fevereiro alterou-se a expressão de forma a que esta encontrasse um comentário correspondente ao mês de junho.

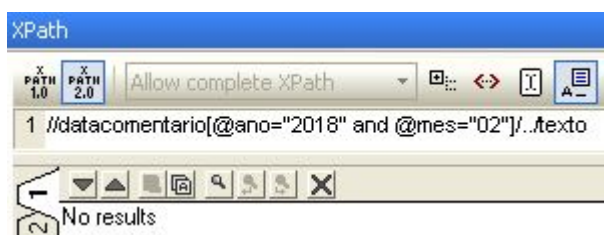
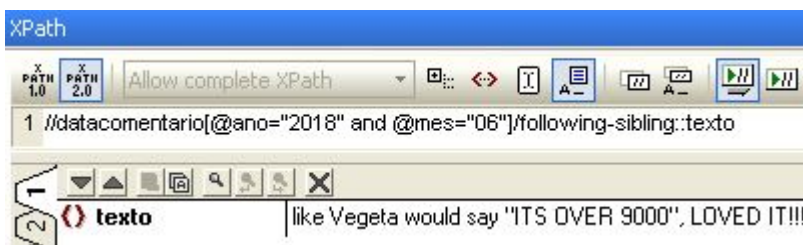


Figura 8 - À esquerda a expressão que permite obter todos os comentários efetuados no mês de fevereiro. Abaixo esta expressão alterada de forma a mostrar um resultado compatível com os comentários existentes



ix. Nomes dos utilizadores que fizeram comentários num vídeo com um dado título

Para obter os nomes dos utilizadores que fizeram comentários num vídeo com um dado título recorreu-se a uma expressão que identificasse os comentários que um dado vídeo tinha e de seguida fornecesse o “id” dos utilizadores que fizeram esses comentários, para tal criou-se então a expressão “//comentário[@Titulo="..."]/../@id”. Por último era necessário identificar o utilizador a quem o “id” obtido correspondia, obtendo então o *output* relativo ao atributo “username” desse utilizador, pelo que então a expressão final obtida foi “//utilizador[@id=(//comentário[@Titulo="..."]/../@id)]/@username”.

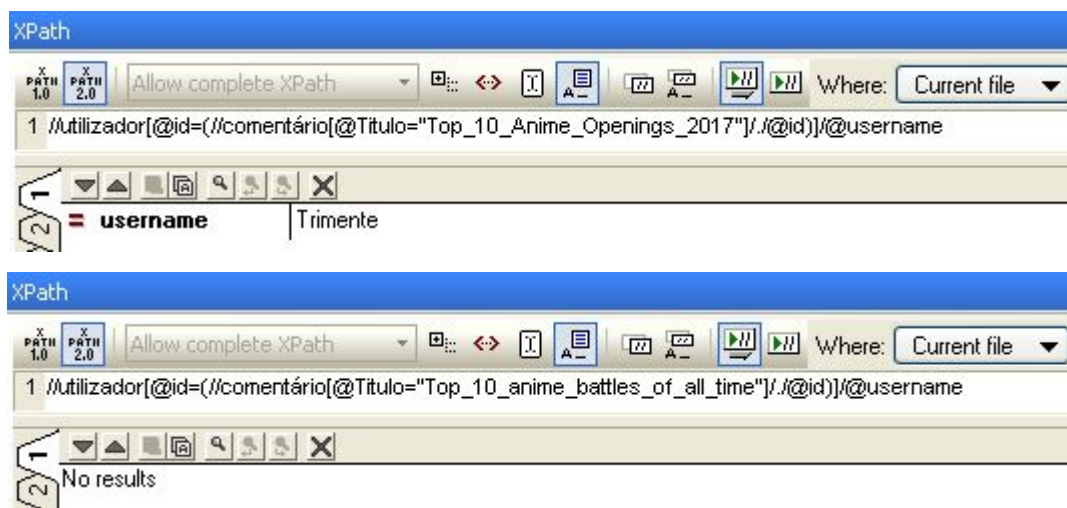


Figura 9 - Acima os nomes dos utilizadores que fizeram comentários num dado vídeo, no primeiro caso o vídeo em questão tinha um comentário, no segundo caso o vídeo não tinha comentários

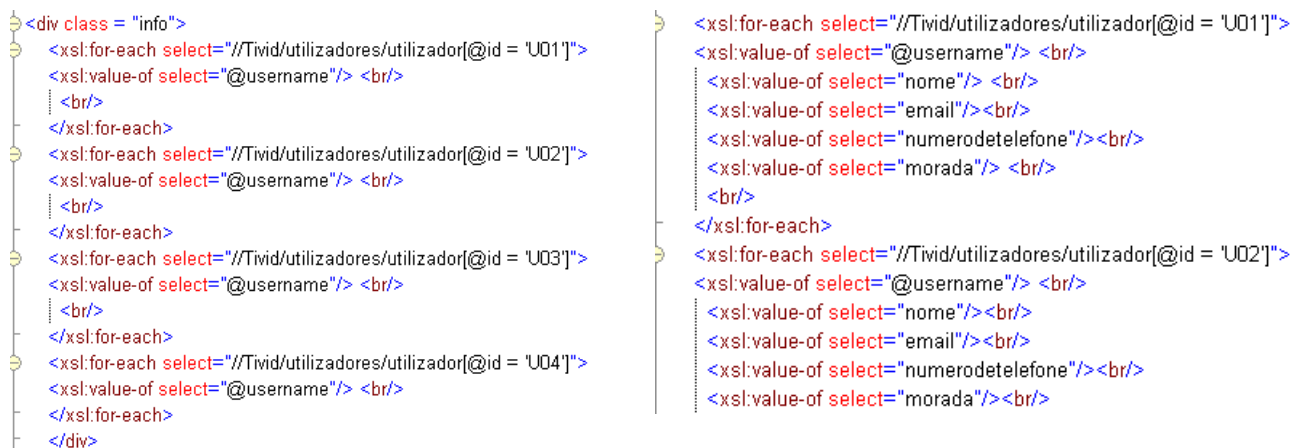
3. Transformações XSLT

Para este grupo, tinha-se como objetivo converter o documento XML usado do primeiro trabalho prático, e usar a linguagem XSLT para transformar numa página XHTML. Como objetivo inicial recorreu-se ao uso da página “home” criada no trabalho prático anterior. De seguida para a realização dos outros tópicos recorreu-se à linguagem XSLT. Para cada um dos tópicos criados (utilizadores, vídeos e playlists), foram criadas duas páginas, uma com informação média e outra com informação mais detalhada.

i. XML para XHTML

i. Utilizadores

Começando com o primeiro tópico, os utilizadores. Para a execução da tarefa, foi usado um for-each. Esta função do XSLT permite aceder a qualquer “nod” no nosso documento XML, permitindo ir buscar a informação que é necessária. Dentro do for-each, foi colocada uma expressão XPATH, na qual, consiste em filtrar a respetiva informação que queremos adicionar. Dentro do for-each, recorreu-se a outra função do XSLT chamada de value-of no qual permite ir buscar o valor contido dentro do for-each, quer sejam elementos, quer sejam elementos contendo apenas texto.



```

<div class = "info">
  <xsl:for-each select="//Tivid/usuarios/utilizador[@id = 'U01']">
    <xsl:value-of select="@username"/> <br/>
  </xsl:for-each>
  <xsl:for-each select="//Tivid/usuarios/utilizador[@id = 'U02']">
    <xsl:value-of select="@username"/> <br/>
  </xsl:for-each>
  <xsl:for-each select="//Tivid/usuarios/utilizador[@id = 'U03']">
    <xsl:value-of select="@username"/> <br/>
  </xsl:for-each>
  <xsl:for-each select="//Tivid/usuarios/utilizador[@id = 'U04']">
    <xsl:value-of select="@username"/> <br/>
  </xsl:for-each>
</div>

<xsl:for-each select="//Tivid/usuarios/utilizador[@id = 'U01']">
  <xsl:value-of select="@username"/> <br/>
  <xsl:value-of select="nome"/> <br/>
  <xsl:value-of select="email"/> <br/>
  <xsl:value-of select="numerosdetelefone"/> <br/>
  <xsl:value-of select="morada"/> <br/>
</xsl:for-each>
<xsl:for-each select="//Tivid/usuarios/utilizador[@id = 'U02']">
  <xsl:value-of select="@username"/> <br/>
  <xsl:value-of select="nome"/> <br/>
  <xsl:value-of select="email"/> <br/>
  <xsl:value-of select="numerosdetelefone"/> <br/>
  <xsl:value-of select="morada"/> <br/>

```

Figura 10 - À esquerda o código XSLT usado para gerar a página XHTML com informação média dos utilizadores. À direita a informação detalhada.

ii. Vídeos

Passando para o próximo tópico, os vídeos. O processo usado é idêntico da maneira de como foi feita os utilizadores, usando um for-each. Esta função do XSLT permite aceder a qualquer “nod” no nosso documento XML, permitindo ir buscar a informação que é necessária. Dentro do for-each, foi colocada uma expressão XPATH, na qual, consiste em filtrar a respetiva informação que queremos adicionar. Dentro do for-each, recorreu-se a outra função do XSLT chamada de value-of no qual permite ir buscar o valor contido dentro do for-each, quer sejam elementos, quer sejam elementos contendo apenas texto.

```

<xsl:for-each select="//videos/video[@Titulo = 'Top_10_anime_battles_of_all_time']">
  <xsl:value-of select="@Titulo"/> <br/>
</xsl:for-each>
<xsl:for-each select="//videos/video[@Titulo = 'Top_10_Anime_Openings_2017']">
  <xsl:value-of select="@Titulo"/> <br/>
</xsl:for-each>
<xsl:for-each select="//videos/video[@Titulo = 'OMG_I_found_RYAN_REYNOLDS_in_PORTUGAL_MUST_WATCH']">
  <xsl:value-of select="@Titulo"/> <br/>
</xsl:for-each>
<xsl:for-each select="//videos/video[@Titulo = 'I_GLUED_MY_BROTHER_TO_THE_FLOOR']">
  <xsl:value-of select="@Titulo"/> <br/>
</xsl:for-each>

<xsl:for-each select="//videos/video[@Titulo = 'Top_10_anime_battles_of_all_time']">
  <xsl:value-of select="@Titulo"/> <br/>
  <xsl:value-of select="@des"/> <br/>
  <xsl:value-of select="duração"/> <br/>
</xsl:for-each>
<xsl:for-each select="//videos/video[@Titulo = 'Top_10_Anime_Openings_2017']">
  <xsl:value-of select="@Titulo"/> <br/>
  <xsl:value-of select="@des"/> <br/>
  <xsl:value-of select="duração"/> <br/>
</xsl:for-each>

```

Figura 11 - Acima observa-se o código XSLT usado para gerar a página com informação média acerca dos vídeos presentes no website. À esquerda o código XSLT para a informação detalhada de cada vídeo

iii. Playlists

Passando para o último tópico, as playlists. O processo usado é idêntico da maneira de como foi feita os utilizadores e os vídeos, usando um for-each. Esta função do XSLT permite aceder a qualquer “nod” no nosso documento XML, permitindo ir buscar a informação que é necessária. Dentro do for-each, foi colocada uma expressão XPATH, na qual, consiste em filtrar a respetiva informação que queremos adicionar. Dentro do for-each, recorreu-se a outra função do XSLT chamada de value-of no qual permite ir buscar o valor contido dentro do for-each, quer sejam elementos, quer sejam elementos contendo apenas texto.

```
<xsl:for-each select="//Playlists/playlist[@TT='Pra_ver']">
<a href="">Pra Ver</a> <br/>
<xsl:value-of select="descricao"/> <br/>
</xsl:for-each>
<a href="">Playlist Frescura</a><br/>
<xsl:for-each select="//Playlists/playlist[@TT='Playlist_frescura']">
<xsl:value-of select="descricao"/> <br/>
| <br/>
</xsl:for-each>
</div>
```

Figura 12 - À esquerda observamos o código XSLT com a informação média à cerca das playlists. Abaixo temos o código XSLT com informação detalhada relativa a cada uma das playlists

```
<div class = "info">
<xsl:for-each select="//Tvid/Playlists/playlist[@ip = 'p_2']/vid[1]">
<xsl:value-of select="@V"/> <br/>
<br/>
<xsl:for-each select="//Tvid/Playlists/playlist[@ip = 'p_2']/vid[2]">
<xsl:value-of select="@V"/> <br/>
- </xsl:for-each>
| <br/>
- </div>
```

```
<div class = "info">
<xsl:for-each select="//Tvid/Playlists/playlist[@ip = 'p_1']/vid[1]">
<xsl:value-of select="@V"/> <br/>
<br/>
<xsl:for-each select="//Tvid/Playlists/playlist[@ip = 'p_1']/vid[2]">
<xsl:value-of select="@V"/> <br/>
- </xsl:for-each>
| <br/>
- </div>
```

ii. XML para XML

Para esta alínea era pedido que fosse feito um XSLT, e que no final, gerasse um documento XML, e ao mesmo tempo validado por um documento DTD, fornecido no enunciado. Para executar a tarefa, o grupo teve que olhar com atenção sobre o documento DTD apresentado. Reparamos que o user, no documento XML vai consistir em mais do que um “user”, pois o símbolo “*”, corresponde ao facto de existir um ou mais “users” no documento XML, (assim como outros elementos que vão estar presentes no documento), assim como o elemento “publishedVideo”. Com a leitura do DTD feita passou-se para a construção do XSLT para gerar o documento XML.

Como muitos dos elementos são atributos, definiu-se, primeiro, dois atributos com o nome de “user1” e “user2”, usando a linguagem do XSLT “param name”, no qual cria um parâmetro com o respectivo “name”. De seguida passou-se para a construção dos elementos que vão fazer parte do nosso XML. Começando então com o elemento “XMLTubeUserInfo”, pois vai ser o elemento principal que vai constar o user e o publishedVideo. Depois disso, passou-se a informação do “user”.

Começando por atribuir um atributo de nome “username”, e de seguida, recorreu-se ao “value-of “, no qual permiti retornar o valor do nó retornado, neste caso, quando formos a retornar o valor desse nó, associa-se esse nó ao atributo “username”. O processo usado para fazer “birthdate” e “numberOfComments” foi idêntico para fazer o “username” a única coisa que se alterou foi o respetivo atributo associado. Com a informação do user tratada, passou-se para a associação do vídeo a esse utilizador. Neste caso como queremos percorrer o nó com os elementos do primeiro “user” recorreu-se a uma “for-each” no qual a expressão XPath, vai corresponder à associação do vídeo e o id do utilizador. Finalmente passou-se para a construção dos elementos que correspondem ao elemento “publishedVideo”. O processo usado também foi idêntico

Na construção dos elementos que continham a informação dos users. Criando um atributo “id” no qual vai estar relacionado com o “id” do vídeo, sendo possível a associação deste atributo com o atributo do “user”. A seguir retorna-se os valores dos nós do atributo “id”. Sendo o “título” e “videoLink” como PCDATA, retornou-se apenas o valor do nó. No final criou-se mais um utilizador com o atributo “user2” seguindo o mesmo raciocínio do primeiro utilizador criado.

```
<xsl:param name="user1">JohnWick98</xsl:param>
<xsl:param name="user2">CopyCat101</xsl:param>
<xsl:template match="/">
  <XMLTubeUserInfo>
    <user>
      <xsl:attribute name="username">JohnWick98<xsl:value-of select="//user[@id=$user1]/attribute::username"></xsl:value-of></xsl:attribute>
      <xsl:attribute name="birthdate">08/01/1998<xsl:value-of select="//user[@id=$user1]/attribute::birthdate"></xsl:value-of></xsl:attribute>
      <xsl:attribute name="numberOfComments">6<xsl:value-of select="//user[@id=$user1]/attribute::numberOfComments"></xsl:value-of></xsl:attribute>
      <name>Luis<xsl:value-of select="//user[@id=$user1]/name"></xsl:value-of></name>
      <xsl:for-each select="publishedVideo[Video/@id = //user[@id=$user1]]"></xsl:for-each>
        <publishedVideo>
          <xsl:attribute name="id">v1<xsl:value-of select="@id"></xsl:value-of></xsl:attribute>
          <titulo>Top 10 places to go on the new year<xsl:value-of select="titulo"></xsl:value-of></titulo>
          <videoLink>Error 404... link not found...<xsl:value-of select="videoLink"></xsl:value-of></videoLink>
        </publishedVideo>
      </user>
    <user>
      <xsl:attribute name="username">CopyCat101<xsl:value-of select="//user[@id=$user2]/attribute::username"></xsl:value-of></xsl:attribute>
      <xsl:attribute name="birthdate">01/08/2000<xsl:value-of select="//user[@id=$user2]/attribute::birthdate"></xsl:value-of></xsl:attribute>
      <xsl:attribute name="numberOfComments">4<xsl:value-of select="//user[@id=$user2]/attribute::numberOfComments"></xsl:value-of></xsl:attribute>
      <name>Carlos<xsl:value-of select="//user[@id=$user2]/name"></xsl:value-of></name>
      <xsl:for-each select="publishedVideo[Video/@id = //user[@id=$user2]]"></xsl:for-each>
        <publishedVideo>
          <xsl:attribute name="id">v2<xsl:value-of select="@id"></xsl:value-of></xsl:attribute>
          <titulo>Fallout76 is a good game, and wheres why<xsl:value-of select="titulo"></xsl:value-of></titulo>
          <videoLink>Error 404... link not found...<xsl:value-of select="videoLink"></xsl:value-of></videoLink>
        </publishedVideo>
      </user>
    </XMLTubeUserInfo>
  </xsl:template>
```

Figura 13 – Possível resolução de um documento XSLT para gerar um documento XML

```
<!DOCTYPE XMLTubeUserInfo SYSTEM "XMLTubeUserInfo.dtd">
<XMLTubeUserInfo>
  <user username="JohnWick98" birthdate="08/01/1998" numberOfComments="6">
    <name>Luis</name>
    <publishedVideo id="v1">
      <titulo>Top 10 places to go on the new year</titulo>
      <videoLink>Error 404... link not found...</videoLink>
    </publishedVideo>
  </user>
  <user username="CopyCat101" birthdate="01/08/2000" numberOfComments="4">
    <name>Carlos</name>
    <publishedVideo id="v2">
      <titulo>Fallout76 is a good game, and wheres why</titulo>
      <videoLink>Error 404... link not found...</videoLink>
    </publishedVideo>
  </user>
</XMLTubeUserInfo>
```

Figura 14 - XML gerado através do documento XSLT criado

File E:\TP3_43033_45125_45156(2)\GRUPO2_XMLTOXML\XMLTubeUserInfo.xml is valid.

Figura 15 - Comprovação da validação do documento XML

```
<!ELEMENT XMLTubeUserInfo (user*)>
<!ELEMENT user (name, publishedVideo*)>
<!ATTLIST user username ID #REQUIRED>
<!ATTLIST user birthdate CDATA #IMPLIED>
<!ATTLIST user numberOfComments CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT publishedVideo (titulo, videoLink)>
<!ATTLIST publishedVideo id ID #REQUIRED>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT videoLink (#PCDATA)>
```

Figura 16 - Documento DTD usado para a validação do documento XML

4. Framework CSS

Foi-nos pedido para utilizar também a linguagem CSS para formatar e "embelezar" o trabalho. Para isso aproveitamos conhecimentos adquiridos para a realização do trabalho prático 2 e aplicámo-los neste terceiro trabalho prático.

5. Vantagens e Desvantagens da solução apresentada

O XPath na verdade não tem exatamente vantagens e desvantagens concretamente bem definidas, no entanto, no caso das expressões utilizadas neste trabalho é que são muito generalizadas, o que, apesar de apresentar a vantagem de que qualquer alteração ao XML e DTD não invalida as expressões (desde que o conteúdo contido nas mesmas não seja alterado), como é usada uma forma de procura que primeiro verifica todos os nós, no caso de um ficheiro XML de grandes dimensões estas expressões levariam muito mais tempo a obter resultados.

Como qualquer linguagem tem as suas vantagens e desvantagens, o XSLT não é exceção. Ao longo do trabalho o grupo deparou-se que a linguagem XSLT é vantajosa para especificar: uma estrutura XML noutra estrutura XML e filtros sobre os componentes a transformar, nomeadamente qualquer ficheiro XML. Em relação às desvantagens do XSLT, é uma linguagem que não permite especificar: de modo preciso a paginação de texto em páginas, o tratamento preciso de grafismo em texto, e a formatação, paginação e grafismo orientado ao suporte à tipografia.

6. Conclusões

Com a realização deste trabalho o grupo conseguiu consolidar os conhecimentos obtidos na disciplina relativamente ao XPath e ao XSLT, tendo até conseguido entender certos conceitos que previamente não tinham sido bem conseguidos. No entanto neste trabalho também existiram adversidades, especialmente no grupo relativo ao XPath. Algumas das expressões apresentadas não são as mais corretas possíveis, no entanto o grupo encontrou problemas ao usar certos termos e certas

funções do XPath, tendo mesmo desistido de as usar em algumas ocasiões e tendo acabado por adotar um método diferente para obter a resolução do problema. Noutras ocasiões o grupo divergiu na interpretação do conteúdo do enunciado, tendo achado certos pontos um pouco confusos, o que levou o grupo a apresentar a solução que consideraram mais acertada (como foi exemplo da expressão “Título do último vídeo publicado por um utilizador”, em que o grupo se dividiu na opinião que seria o último vídeo relativamente à data da sua publicação ou o último vídeo em relação à posição dos seus nós, tendo optado pela última). Houve ainda problemas na utilização da função “dateTime ()”, tal como referido anteriormente, que apesar da expressão obtida aparentar resultar da forma pretendida, inicialmente apresentava o erro “Too many items” e após alterar ligeiramente a expressão para melhor corresponder à função, passou a apresentar erros relativamente ao “xs:time”, pelo que o grupo optou por desistir dessa expressão e fazer alterações aos ficheiros de forma a poder criar outra expressão que apresentasse os resultados pretendidos.

Ainda assim, apesar das adversidades o grupo considera que os resultados obtidos foram satisfatórios e as transformações XSLT obtidas foram as pretendidas, pelo que consideramos a generalidade do trabalho um sucesso.