

Bruno Marra (3029)  
Daniel Freitas (2304)  
Gustavo Viegas (3026)  
Vitor Luís (3045)

## **Trabalho Prático 3**

Documentação de Trabalho Prático - TP3

Universidade Federal de Viçosa - Campus Florestal  
Gestão, Recuperação e Análise de Informações  
Ciência da Computação

Florestal  
4 de Novembro de 2019

# **Lista de ilustrações**

Figura 1 – Fluxo de envios e recebimentos do Crawler.	10
Figura 2 – Exemplo de execução do crawler.	11
Figura 3 – Modelo relacional para dados de filmes.	22
Figura 4 – Inserção após validação.	26
Figura 5 – Tela inicial do sistema.	32
Figura 6 – Tela de adição de séries/filmes.	33
Figura 7 – Tela de busca por Terry Crews.	34
Figura 8 – Tela de busca por Séries em espanhol.	34
Figura 9 – Tela de busca pela série do personagem Chris Rock OU Todo Mundo Odeia o Chris	35
Figura 10 – Tela de busca por séries de comédia	36

# Sumário

<b>1</b>	<b>Introdução</b>	<b>4</b>
<b>2</b>	<b>Ambiente</b>	<b>5</b>
2.1	Subindo a aplicação	5
<b>3</b>	<b>Tecnologias</b>	<b>6</b>
3.1	Frontend	6
3.2	Backend	6
3.3	Fontes de Dados	7
<b>4</b>	<b>Upload de arquivos XML</b>	<b>8</b>
<b>5</b>	<b>Crawler</b>	<b>9</b>
<b>6</b>	<b>Definição de esquemas XML</b>	<b>17</b>
<b>7</b>	<b>As bases de dados</b>	<b>22</b>
7.1	Banco de dados relacional - MySQL	22
7.2	Banco de dados não relacional - MongoDB	23
7.3	Carga de dados	24
<b>8</b>	<b>Validações e inserções nas bases de dados</b>	<b>25</b>
<b>9</b>	<b>Estratégias de recuperação</b>	<b>29</b>
9.1	Séries	30
9.2	Filmes	30
<b>10</b>	<b>Execução do sistema</b>	<b>32</b>
<b>11</b>	<b>Conclusão</b>	<b>37</b>
	<b>Referências</b>	<b>38</b>

# 1 Introdução

Neste trabalho é apresentado um sistema de recuperação de informação (RI) cujo contexto é o mesmo do primeiro trabalho prático desta disciplina. Assim, este trabalho é uma expansão do projeto anterior, com foco no contexto de RI. As tecnologias utilizadas anteriormente foram mantidas.

O sistema é uma plataforma de consulta de dados de filmes e séries, obtidos originalmente por APIs e websites públicos, que estão armazenados em diversas origens. Contemplando a nova especificação, o sistema expandiu-se para permitir a importação de novos filmes e séries através de arquivos XML. Além disso, a ferramenta de pesquisa foi refinada, disponibilizando melhores resultados para a busca do usuário.

Neste documento serão apresentadas como as importações, validações dos arquivos importados e inserções desses dados nas bases de dados foram realizadas. Ainda, serão mostrados como os esquemas XML foram definidos, como as pesquisas podem ser realizadas e o Crawler, evolução do Proto-Crawler<sup>1</sup> do primeiro trabalho prático.

---

<sup>1</sup> O Proto-Crawler era formado por scripts sem correspondências entre si, que eram executados de forma isolada nas páginas web. A junção dos conteúdos era realizada manualmente.

## 2 Ambiente

Para o desenvolvimento do trabalho foram utilizados 4 containers principais que utilizam das imagens a seguir:

- **MySQL** - Tag *5.7* da imagem oficial (do *DockerHub*) do banco de dados relacional MySQL.
- **Mongo** - Tag *4.2* da imagem oficial (do *DockerHub*) do banco de dados não relacional MongoDB.
- **Elixir** - Tag *latest* da imagem oficial (do *DockerHub*) que foi extendida em um *Dockerfile* próprio, na raiz da pasta **api** no projeto.
- **Node** - Tag *12* da imagem oficial (do *DockerHub*) que foi extendida em um *Dockerfile* próprio, na raiz da pasta **front** no projeto.

Os containers foram orquestrados com o *Docker Compose*, e suas configurações de ambiente, portas, network e etc. podem ser consultadas no arquivo **docker-compose.yml** na raiz do projeto.

### 2.1 Subindo a aplicação

Para subir a aplicação, em qualquer máquina com o *Docker* e *Docker Compose* devidamente instalados, basta executar no terminal, na raiz do projeto:

```
1 docker-compose up
```

A API estará escutando em <http://localhost:4000>, e o frontend da aplicação estará disponível em <http://localhost>.

# 3 Tecnologias

Para a realização deste trabalho diversas tecnologias foram abordadas. Elas são descritas neste capítulo de acordo com o seu uso.

## 3.1 Frontend

O frontend da aplicação é um cliente *web* com uma stack HTML/JS/CSS padrão, mas com o uso de alguns frameworks, pré-processadores e bibliotecas para auxílio, roteamento, definição de componentes reutilizáveis, etc. As principais são:

- [VueJS](#) - Framework JavaScript.
- [Apollo](#) - Cliente GraphQL em JavaScript.
- [SASS](#) - Pré-processador CSS.

## 3.2 Backend

O backend da aplicação é um web service HTTP que serve os dados a partir de um endpoint. Com o uso do GraphQL, a API desenvolvida oferece os schemas para que o cliente possa consumir os dados. Para o desenvolvimento do backend, as tecnologias relevantes utilizadas foram:

- [Elixir](#) - Linguagem de programação funcional, que executa na máquina virtual *BEAM*.
- [Phoenix](#) - Framework web para Elixir.
- [Absinthe](#) - Plugin de definição de schemas e servidor GraphQL para Elixir. Também oferece o *GraphiQL*, que será abordado em capítulos posteriores.
- [Ecto](#) - *Toolkit* de mapeamento de dados e abstração de linguagem SQL em bancos de dados relacionais, para Elixir.
- [NodeJS](#) - Linguagem de programação script, usada pelo crawler de busca.

### 3.3 Fontes de Dados

A aplicação faz uso de três fontes de dados principais, sendo duas delas bancos de dados estruturados pelos autores do trabalho, e a outra uma API externa. Elas são:

- MySQL - Banco de dados relacional.
- MongoDB - Banco de dados não-relacional.
- OMDB - OpenMovie Database: uma API RESTful com dados públicos sobre filmes e séries.

## 4 Upload de arquivos XML

Uma das funcionalidades adicionadas neste trabalho prático foi a possibilidade do usuário realizar o *upload* de um filme ou uma série através de um arquivo XML.

Basicamente, como existe a interação do usuário com o sistema, algumas mudanças foram necessárias no *frontend*. Com o auxílio do *framework* Vue, dois novos botões foram acrescentados: um para adicionar um filme e outro para uma série. Na codificação do próprio botão, foi possível limitar a seleção de arquivos que não estivessem com a extensão ".xml". Entretanto, como já se sabe, a extensão de um arquivo pouco importa nas mãos de um usuário que deseja realizar uma inserção inválida no banco de dados do sistema ou que simplesmente salvou o arquivo com a extensão errada. Por conta disso, uma validação mais rigorosa é feita posteriormente ([Capítulo 8](#)).

Contudo, para que a validação seja feita, é necessário disponibilizar para o *backend* o arquivo selecionado pelo usuário. Assim, ao clicar no botão, uma requisição *http* é realizada com o método POST para um caminho predefinido pelo grupo. Enquanto o *frontend* aguarda uma resposta, o *backend* recebe o novo XML e verifica se ele é válido. Em caso afirmativo, o retorno recebido será igual à 201, ou seja, a execução do método foi um sucesso. Caso contrário, o retorno será 409, significando que ocorreu um conflito. Logo, com base no retorno recebido pelo *frontend*, o usuário é avisado com uma mensagem.

Além dessas duas formas de inserção, criou-se um *crawler* para funcionar como um cliente que envia séries à API. Devido a sua complexidade, ele encontra-se detalhado no [Capítulo 5](#).

## 5 Crawler

Esta seção descreve o Crawler criado para gerar arquivos XML válidos para inserção.

O Crawler foi concebido apenas para o contexto das séries em inglês. Ele visita o OMDBAPI ([OMDBAPI, 2014](#)) para extrair a maior parte dos dados e o IMDB ([IMDB, 1990](#)) para extrair os dados referentes aos atores e à classificação indicativa das séries.

A decisão de manter o Crawler apenas para o contexto das séries em inglês se deve ao fato de que a utilização de uma base de dados não relacional, como é o caso do MongoDB,

O Crawler foi construído utilizando-se Node JS, um interpretador de JavaScript assíncrono com código aberto orientado a eventos ([JOYENT INC., 2009](#)). Ele foi projetado para construir um arquivo XML válido para inserção contendo registros de uma ou mais séries durante a busca. Ele se baseia nos scripts que formavam o **Proto-Crawler** (não era um crawler, de fato) utilizado no primeiro trabalho prático desta disciplina. Tais scripts eram isolados e deveriam ser executados nos browsers nas respectivas páginas web para extração dos dados. O Crawler criado visa otimizar todo este processo, realizando a busca automatizada e a junção automática dos dados coletados. Para tanto, tais scripts foram reformulados e uma interface de busca foi definida.

É interessante ressaltar que dados que apenas nossa API fornece, como a indicação dos protagonistas e os *spin-offs* associados àquela série, não puderam ser identificados pelo Crawler. Isto se deve a dois fatores complicativos: para indicar os protagonistas, o grupo havia definido que eram protagonistas aqueles personagens/atores que participaram de todos os episódios ou que sua participação era muito impactante na série; para identificar os *spin-offs* era necessário uma busca na web, que gerava resultados às vezes incompletos ou difíceis de serem interpretados. No primeiro caso, o IMDb não fornece para todas as séries as informações que possibilitam essa identificação; no segundo caso, não haveria uma forma de saber quando parar de realizar as buscas e, além disso, seria bastante complicado capturar os dados relevantes das páginas que levavam à identificação de *spin-offs*.

Abaixo na [Figura 1](#) é possível visualizar como se dá o fluxo de envios e recebimentos do Crawler para com as fontes dos dados.



Figura 1 – Fluxo de envios e recebimentos do Crawler.

Em suma, inicialmente o Crawler percorre a lista de séries do IMDb (1) e captura o ID de uma das séries (2). Em seguida, faz uma requisição ao OMDb API (3) que retorna a maior parte das informações das séries (título, escritores, idiomas, duração, etc.) (4). Ainda, o Crawler faz uma requisição ao IMDb (1) que retorna os atores e a classificação indicativa (2).

A seguir serão mostrados alguns dos principais trechos de código que dizem respeito ao Crawler. Os códigos que complementam tais trechos se encontram em anexo a este trabalho. Para habilitar o Crawler, basta executar o seguinte comando na raiz do projeto:

```
1 docker-compose up crawler
```

A [Figura 2](#) exemplifica a execução do crawler, que mostra no log os autores recuperados no script, como exemplo.

```

~/Projects/gradi master +1 !1 > docker-compose run crawler          02:07:18
yarn install v1.17.3
warning package-lock.json found. Your project contains lock files generated by tools other than Yarn. It is advised not
  to mix package managers in order to avoid resolution inconsistencies caused by unsynchronized lock files. To clear thi
s warning, remove package-lock.json.
[1/4] Resolving packages...
success Already up-to-date.
Done in 0.23s.
yarn run v1.17.3
$ yarn && node series/crawler.js
warning package-lock.json found. Your project contains lock files generated by tools other than Yarn. It is advised not
  to mix package managers in order to avoid resolution inconsistencies caused by unsynchronized lock files. To clear thi
s warning, remove package-lock.json.
[1/4] Resolving packages...
success Already up-to-date.
--- INICIANDO ---
https://www.imdb.com/search/keyword/?keywords=computer&ref_=kw_ref_typ&sort=num_votes,desc&mode=detail&title_type=tvSer
ies&page=1
    Pegando do OMDB: http://www.omdbapi.com/?i=tt1119644&apikey=6f5f8dc6
    Pegando do OMDB: http://www.omdbapi.com/?i=tt0149460&apikey=6f5f8dc6
String do actors:      <actor character="Peter Bishop">Anna Torv</actor>
<actor character="Astrid Farnsworth">Jasika Nicole</actor>
<actor character="Dr. Walter Bishop">John Noble</actor>
<actor character="Phillip Broyles">Lance Reddick</actor>
<actor character="Nina Sharp">Blair Brown</actor>
<actor character="The Observer">Michael Cerveris</actor>
<actor character="Charlie Francis">Kirk Acevedo</actor>
<actor character="Lincoln Lee">Seth Gabel</actor>
<actor character="Brandon Fayette / ...">Ryan McDonald</actor>
<actor character="John Scott">Mark Valley</actor>
<actor character="Captain Windmark">Michael Kopsa</actor>
<actor character="Ella">Lily Pilblad</actor>
<actor character="Dr. William Bell">Leonard Nimoy</actor>
<actor character="Rachel">Ari Graynor</actor>
<actor character="December">Eugene Lipinski</actor>
<actor character="David Robert Jones">Jared Harris</actor>
<actor character="Thomas Jerome Newton">Sebastian Roché</actor>
<actor character="Anil">Shaun Smyth</actor>
<actor character="Sam Weiss">Kevin Corrigan</actor>
<actor character="Peter Bishop">Anna Torv</actor>
<actor character="Astrid Farnsworth">Jasika Nicole</actor>
<actor character="Dr. Walter Bishop">John Noble</actor>
<actor character="Phillip Broyles">Lance Reddick</actor>
<actor character="Nina Sharp">Blair Brown</actor>
<actor character="The Observer">Michael Cerveris</actor>
<actor character="Charlie Francis">Kirk Acevedo</actor>
<actor character="Lincoln Lee">Seth Gabel</actor>
<actor character="Brandon Fayette / ...">Ryan McDonald</actor>
<actor character="John Scott">Mark Valley</actor>
<actor character="Captain Windmark">Michael Kopsa</actor>
<actor character="Ella">Lily Pilblad</actor>
<actor character="Dr. William Bell">Leonard Nimoy</actor>
<actor character="Rachel">Ari Graynor</actor>
<actor character="December">Eugene Lipinski</actor>
<actor character="David Robert Jones">Jared Harris</actor>
<actor character="Thomas Jerome Newton">Sebastian Roché</actor>
<actor character="Anil">Shaun Smyth</actor>
<actor character="Sam Weiss">Kevin Corrigan</actor>

```

Figura 2 – Exemplo de execução do crawler.

O trecho de código abaixo ([Listing 5.1](#)) ilustra a definição do método *crawl*. Os três primeiros valores indicam o link para se realizar a busca pelas séries, a quantidade máxima de séries que serão montadas e a página referente à paginação da busca, respectivamente. Em linhas gerais, este método é responsável por percorrer toda a página que lista as séries (50 séries por página) e armazena os respectivos *imdb IDs* em uma coleção, retornando-a ao método chamador.

```

1 const IMDB_IDS_ENDPOINT = 'https://www.imdb.com/search/keyword/
2 ?keywords=series&ref_=fn_al_kw_1'
3 const quantidade = 2 // numero de series a serem consideradas
4 let imdb_ids_page = 1 // pagina referente a paginacao
5
6 const crawl = async () => {
7   console.log('--- INICIANDO ---')
8   try {
9     console.log(`\$${IMDB_IDS_ENDPOINT}&page=\$${imdb_ids_page}`);
10    // requisicao HTTP:
11    const response = await axios
12      .get(`\$${IMDB_IDS_ENDPOINT}&page=\$${imdb_ids_page}`)
13
14    const $ = che.load(response.data)
15    // percorrendo parte especifica da pagina web
16    // essa parte contem as series e seus ids
17    const imbdids = await $('.lister-item-header')
18      .map((i, element) => {
19        const linkimdb = $(element).find('a').attr('href')
20
21        // /title/tt0087800/?ref_=kw_li_tt
22        let imbdid = linkimdb.substring(7) // cortando /title/
23        // capturando apenas o imdb id
24        imbdid = imbdid.substring(0, imbdid.indexOf('/'))
25        return imbdid // adicionando id a colecao
26      }).get()
27
28    return imbdids
29  } catch (e) {
30    console.error(e)
31    return null
32  }
33}

```

Listing 5.1 – Definição da função *crawl*

O trecho de código abaixo ([Listing 5.2](#)) mostra o início (linha 6) e o fim (linha 18) da construção do arquivo XML válido gerado ao término da execução. Este trecho é o único trecho executado na chamada do módulo Crawler. Ele realiza chamada para os demais métodos e é responsável por decidir quantas séries estarão presentes no arquivo XML ao final (linha 19). O arquivo gerado é enviado para a API logo em seguida (linha

22). O método *getFromOmdb* retorna uma coleção de séries, já adequadas ao formato para inserção no formato XML, sendo essas *strings* concatenadas logo depois (linha 15).

A função invocada na linha 11 é na verdade uma abstração para a função definida em Listing 5.3. Ela é definida no arquivo *crawler.js* e é responsável por realizar uma requisição HTTP e enviar o corpo HTML correspondente.

Ambos os trechos de código apresentados representam parte da interface de busca. A interface de busca é composta por outros métodos (omitidos aqui, mas presentes em anexo) que realizam o fluxo mostrado na Figura 1.

```

1 (async () => {
2   try {
3     const imdbids = await crawl()
4     if (!imdbids) throw new Error('no_ids_found')
5
6     let seriesetstring = '<serieset>\n'
7     // construcao do conjunto de series
8     const serieset = await Promise
9       .all(imdbids.slice(0, quantidade))
10      .map(async (imdbid) => {
11        return getFromOmdb(imdbid)
12      })
13    // para cada serie, concatena-se
14    serieset.forEach((series) => {
15      seriesetstring += `${series}\n`
16    })
17
18    seriesetstring += '</serieset>' // fim
19    fs.writeFileSync('series_instance_generated.xml',
20      seriesetstring) // escrita em arquivo
21    // parte omitida -> envio para form //
22    const response = await axios // enviando para a API
23      .post('http://api:4000/crawler', formData,
24        { headers: formData.getHeaders() })
25    console.log('--- FIM ---')
26  } catch (e) {
27    console.error(`Erro Inesperado: ${e}`)
28  }

```

29 } )()

Listing 5.2 – Trecho de código assíncrono que é executado na chamada do módulo do Crawler

Os scripts criados no primeiro trabalho prático desta disciplina precisaram ser reformulados para que houvesse a interação entre eles e o Crawler. Abaixo partes de cada um deles serão mostradas.

Observe o trecho de código Listing 5.3. Um dos fatores complicativos encontrados foi o fato de que o Node JS não possui o mesmo comportamento de um código em JavaScript executado no *browser*. Isto é, os códigos que utilizavam XPath referenciando um documento de uma página web não eram válidos em Node JS, sendo necessário realizar avaliar outras alternativas. A solução encontrada foi realizar uma requisição HTTP e converter o corpo da página web em XML (linha 14). O *loop* que se inicia na linha 16 é responsável por montar o conjunto de atores no formato XML. Note que os caminhos em XPath foram reaproveitados do primeiro trabalho prático.

```

1 module.exports = {
2   async getActorsFromIMDB (html) {
3     const max = 20 // numero maximo de atores retornados
4     let mult = 1
5     let stringBuilder = ''
6
7     // Parse da pagina web para usar XPath
8     const parsed = parse5.parse(html.toString())
9     const xhtml = xmlser.serializeToString(parsed).trim()
10    .replace(/\\n/g, '').replace(/ \\ /g, ' ')
11    .replace('xmlns="http://www.w3.org/1999/xhtml"
12      xmlns:og="http://ogp.me/ns#"
13      xmlns:fb="http://www.facebook.com/2008/fbml"', '')
14    const document = new DOM().parseFromString(xhtml)
15
16    for (let i = 2; i <= max; i++) {
17      let s1 = xpath.evaluate(`//html/body/div[2]/div/
18        div[2]/descendant::*/tbody/tr[${i * mult}]/
19        td[2]/a//text()`, document, null,
20        XPATH_FIRST_ORDERED_TYPE, null).singleNodeValue
21      if (s1 == null) break
22      s1 = s1.textContent.trim()
23
24      // --- PARTE OMITIDA --- //

```

```

25      // concatenacao de atores
26      stringBuilder += '        <actor character="${s2}">
27      ${s1}</actor>'
28      if (i !== max) {
29          stringBuilder += '\n'
30      }
31  }
32
33  console.log(`String do actors: ${stringBuilder}`)
34
35  return stringBuilder // colecao de atores
36 } // ...

```

Listing 5.3 – Trecho de código referente à função getActorsFromIMDB do arquivo *actors.js*

O trecho de código Listing 5.4 abaixo corresponde à construção das séries em si. O que é interessante de ser mostrado são as linhas 19 e 22 (*classification* e *actors*, respectivamente). Elas descrevem tags que são substituídas por outras etapas do Crawler, pois essas informações não são encontradas de forma completa (*actors*) ou estão ausentes (*classification*) na OMDB API. O Crawler utiliza o IMDb para capturar tais informações e as substituições dessas tags são feitas posteriormente pelos valores adequados. Entretanto, ainda assim algumas séries possuem dados incompletos a respeito dos atores e algumas séries não possuem sua classificação indicativa no IMDb. Para tratar isso, valores *default* foram incluídos na ausência de determinados valores.

```

1 module.exports = {
2   getFromOmdb (param) {
3     const jo = (typeof param === 'string')
4       ? JSON.parse(param) : param // JSONObject
5     // capturando os dados individuais da serie
6     let releaseDate = jo.Released.split(',')
7     const title = jo.Title
8     const runtime = jo.Runtime
9     const genres = '        <genre>${
10       jo.Genre
11       .replace(/\s/g, '</genre>\n        <genre>')}</genre>\n'
12     // --- PARTE OMITIDA --- //
13     // construcao da serie
14     return `        <series>
15           <imdbid>${imdbId}</imdbid>

```

```
16   <title>${ title }</title>
17   <release_date>${ releaseDate }</release_date>
18   <runtime>${ runtime }</runtime>
19   <classification>@CLASSIFICATION</classification>
20   <genres>\n${ genres }      </genres>
21   <writers>\n${ writers }      </writers>
22   <actors>\n@ACTORS\n      </actors>
23   <description>${ description }</description>
24   <languages>\n${ languages }      </languages>
25   <countries>\n${ countries }      </countries>
26   <poster>${ poster }</poster>
27   <seasons>${ seasons || 1 }</seasons>
28   </series >‘
29 }
```

Listing 5.4 – Trecho de código referente à função `getFromOmdb` do arquivo `omdb.js`

## 6 Definição de esquemas XML

Esta seção trata da definição de esquemas para validação de arquivos XML.

Diante do contexto definido neste trabalho, foi necessário definir esquemas XML para a inserção de documentos válidos para popular as bases de dados internas. Tais esquemas definem restrições para a organização hierárquica e conteúdo dos documentos XML no contexto de filmes e séries.

Para a definição de esquemas, o *XML Schema Definition* (XSD) foi utilizado. Foram definidos quatro esquemas, sendo dois para filmes e dois para séries, permitindo a inserção de documentos XML com tags e valores em inglês e em português em ambas as abordagens. Cada esquema permite que mais de uma série ou filme estejam presentes em um mesmo arquivo. As principais referências bibliográficas consultadas para a construção dos esquemas se referem às documentações oficiais: ([W3C, 2001c](#)), ([W3C, 2001a](#)) e ([W3C, 2001b](#)).

Abaixo segue um trecho da definição de um arquivo XSD para séries, em inglês ([Listing 6.1](#)). O documento completo em questão pode ser encontrado em anexo a este trabalho.

O que é interessante de ser observado em [Listing 6.1](#) é a característica semelhante à Orientação a Objetos, onde objetos podem ser definidos e referenciados no documento. No trecho de código em questão, observa-se a definição do objeto Serie, que por sua vez referencia demais objetos como Genre, Language, Spinoff, etc. Alguns desses objetos como Genre e Language foram definidos inicialmente para serem enumerações, de modo a filtrar os valores e aumentar a confiabilidade da consistência dos documentos XML recebidos pela aplicação. Entretanto, como documentos gerados pelo Crawler poderiam ser inválidos devido a enumerações não consideradas, o grupo optou por definir tais objetos possuindo apenas um campo *string* para o contexto de séries em inglês, apenas. Os outros três esquemas definidos seguem com enumerações. Note ainda que mais de uma série pode estar presente em um mesmo arquivo, pois o esquema define um conjunto de séries (*seriesset*).

Vale ressaltar que a API fornece a funcionalidade de testes automatizados para confirmar o funcionamento dos arquivos XSD. Desta forma, é possível garantir que eles foram bem definidos e irão atender bem à aplicação. O código apresentado em [Listing 8.1](#) no [Capítulo 8](#) é utilizado para estes testes. Em linhas gerais, arquivos XML de testes são obtidos do diretório *test/resources* e validados para cada um dos esquemas definidos. Tais arquivos de teste constituem arquivos válidos e inválidos (ausência de tags, valores inesperados, etc.).

```

1 <xs:element name="seriesset">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="series" type="Serie" minOccurs="1"
5         maxOccurs="unbounded" />
6     </xs:sequence>
7   </xs:complexType>
8 </xs:element>
9
10 <xs:complexType name="Serie">
11   <xs:all>
12     <xs:element name="imdbid" type="xs:string" />
13     <xs:element name="title" type="xs:string" />
14     <xs:element name="release_date" type="xs:date" />
15     <xs:element name="runtime" type="xs:string" />
16     <xs:element name="classification" type="xs:string" />
17     <xs:element name="genres">
18       <xs:complexType>
19         <xs:sequence>
20           <xs:element name="genre" type="Genre" minOccurs="1"
21             maxOccurs="unbounded" />
22         </xs:sequence>
23       </xs:complexType>
24     </xs:element>
25     <!-- Outros elementos omitidos -->
26   </xs:all>
27 </xs:complexType>
28 <!-- Outras definições omitidas (Genre, Language,
29   Country, Spinoff, ...) -->

```

Listing 6.1 – Trecho da definição de XSD para séries em inglês

Abaixo segue um exemplo de XML que está de acordo com o esquema definido em Listing 6.1. O esquema definido permite que os elementos de uma série estejam em qualquer ordem. As únicas restrições são que os elementos e atributos obrigatórios devem estar presentes no corpo do documento, como é o caso dos campos *imdbid*, *title*, etc.

```

1 <serieset>
2   <series>
```

```
3   <imdbid>tt0035027</imdbid>
4   <title>The Man Who Wouldn't Die</title>
5   <release_date>1942-05-01</release_date>
6   <runtime>65 min</runtime>
7   <classification></classification>
8   <genres>
9     <genre>Crime</genre>
10    <genre>Mystery</genre>
11  </genres>
12  <writers>
13    <writer>Arnaud d'Usseau</writer>
14    <writer>Clayton Rawson</writer>
15    <writer>Brett Halliday</writer>
16  </writers>
17  <actors>
18    <actor character="Michael Shayne">Lloyd Nolan</actor>
19    <actor character="Catherine Wolff">Marjorie Weaver</actor>
20    <actor character="Anna Wolff">Helene Reynolds</actor>
21    <actor character="Dr. Haggard">Henry Wilcoxon</actor>
22    <actor character="Roger Blake">Richard Derr</actor>
23    <actor character="Dudley Wolff">Paul Harvey</actor>
24    <actor character="Phillips – the Butler">
25      Billy Bevan</actor>
26    <actor character="Chief of Police Jonathan
27      Meek">Olin Howland</actor>
28    <actor character="Alfred Dunning">
29      Robert Emmett Keane</actor>
30    <actor character="Zorah Bey">LeRoy Mason</actor>
31    <actor character="Coroner Tim Larsen">Jeff Corey</actor>
32    <actor character="Caretaker">Francis Ford</actor>
33  </actors>
34  <description>A private detective gets mixed up with a
35    phony spiritualist racket.</description>
36  <languages>
37    <language>English</language>
38  </languages>
39  <countries>
40    <country>USA</country>
41  </countries>
```

```

42   <poster>https://m.media-amazon.com/images/M/MV5B0DJjN
43     jI0N2UtZjIwYi00MzliLWI1NDAtMDJlNjc3NTY2YzVkL2ltYWdlXk
44     EyXkFqcGdeQXVyNTM3MDMyMDQ@._V1_SX300.jpg</poster>
45   <seasons>1</seasons>
46 </series>
47 </serieset>
```

Listing 6.2 – XML de acordo com o esquema definido em Listing 6.1

A seguir, Listing 6.3 mostra um XML equivalente ao definido em Listing 6.2, mas com tags e valores em português.

```

1 <séries>
2   <série>
3     <imdbid>tt0035027</imdbid>
4     <título>A Sepultura Vazia</título>
5     <data_lançamento>1942-05-01</data_lançamento>
6     <duração>65 min</duração>
7     <classificação></classificação>
8     <gêneros>
9       <gênero>Crime</gênero>
10      <gênero>Mistério </gênero>
11    </gêneros>
12   <escritores>
13     <escritor>Arnaud d'Usseau</escritor>
14     <escritor>Clayton Rawson</escritor>
15     <escritor>Brett Halliday</escritor>
16   </escritores>
17   <atores>
18     <ator personagem="Michael Shayne">Lloyd Nolan</ator>
19     <ator personagem="Catherine Wolff">Marjorie Weaver</ator>
20     <ator personagem="Anna Wolff">Helene Reynolds</ator>
21     <ator personagem="Dr. Haggard">Henry Wilcoxon</ator>
22     <ator personagem="Roger Blake">Richard Derr</ator>
23     <ator personagem="Dudley Wolff">Paul Harvey</ator>
24     <ator personagem="Phillips - the Butler">
25       Billy Bevan</ator>
26     <ator personagem="Chief of Police Jonathan
27       Meek">Olin Howland</ator>
28     <ator personagem="Alfred Dunning">Robert
```

```
29      Emmett Keane</ator>
30      <ator personagem="Zorah Bey">LeRoy Mason</ator>
31      <ator personagem="Coroner Tim Larsen">Jeff Corey</ator>
32      <ator personagem="Caretaker">Francis Ford</ator>
33  </atores>
34  <descrição>Um detetive particular se confunde com uma
35      falsa festa espiritualista.</descrição>
36  <idiomas>
37      <idioma>Inglês</idioma>
38  </idiomas>
39  <países>
40      <país>USA</país>
41  </países>
42  <pôster>https://m.media-amazon.com/images/M/MV5B0DJjN
43      jI0N2UtZjIwYi00MzliLWI1NDAtMDJlNjc3NTY2YzVkL2ltYWdlXk
44      EyXkFqcGdeQXVyNTM3MDMyMDQ@._V1_SX300.jpg</pôster>
45  <temporadas>1</temporadas>
46  </série>
47 </séries>
```

Listing 6.3 – XML de acordo com um esquema semelhante ao Listing 6.1, que permite tags em português

# 7 As bases de dados

Esta seção trata da modelagem das bases de dados consideradas, bem como das cargas de dados envolvidas. Como as bases de dados foram as mesmas utilizadas no primeiro trabalho prático dessa disciplina, essa seção será mais breve.

As subseções seguintes irão mostrar esses modelos e como foram realizadas as cargas de dados.

## 7.1 Banco de dados relacional - MySQL

O banco de dados relacional foi utilizado para armazenar dados a respeito de filmes. Os dados considerados podem ser visualizados no modelo representado na [Figura 3](#) abaixo.

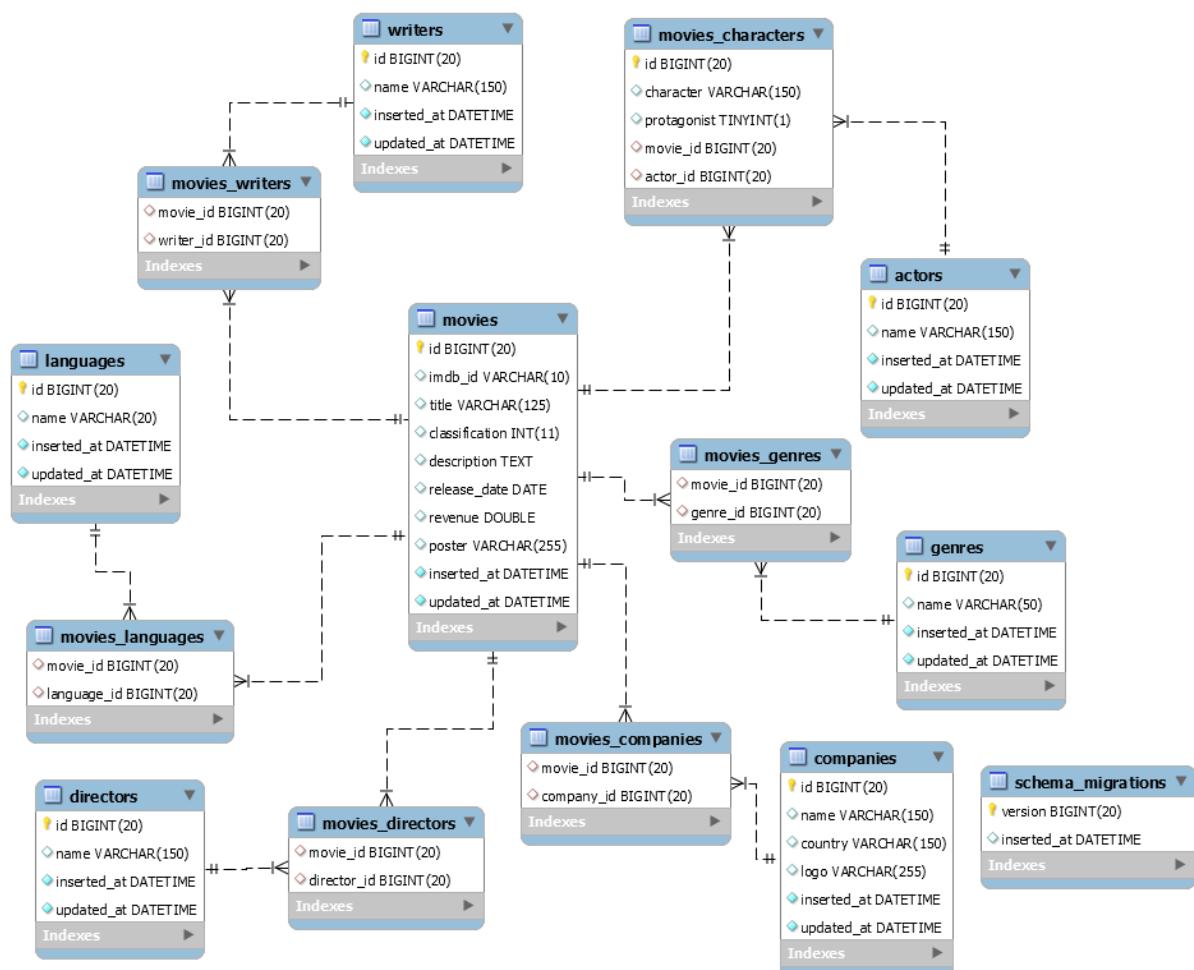


Figura 3 – Modelo relacional para dados de filmes.

## 7.2 Banco de dados não relacional - MongoDB

O banco de dados não-relacional foi utilizado para armazenar dados a respeito de séries de televisão. Os dados considerados podem ser visualizados abaixo no exemplo de inserção da série *Breaking Bad*.

```
# Este exemplo contempla os campos que uma serie pode ter
imdb_id: "tt0903747",
title: "Breaking Bad",
release_date: "2008-01-20",
runtime: "49 min",
classification: "14A",
genres: ["Crime", "Drama", "Thriller"],
writers: ["Vince Gilligan"],
actors: [
    %{name: "Bryan Cranston", character: "Walter White",
        protagonist: true},
    %{name: "Aaron Paul", character: "Jesse Pinkman",
        protagonist: true},
    %{name: "Bob Odenkirk", character: "Saul Goodman"},
    # ... + atores
],
description: "A high school chemistry teacher diagnosed with
    inoperable lung cancer turns to manufacturing and selling
    methamphetamine in order to secure his family's future.",
languages: ["English", "Spanish"],
countries: ["USA"],
poster:
    "https://m.media-amazon.com/images/M/MV5BMjhiMzgxZTctNDc1N
i000TIxLT1hMTYtZTA3ZWfkODRkNmE2XkEyXkFqcGdeQXVyNzkwMjQ5NzM@._
V1_SX300.jpg",
seasons: 5,
spinoffs: [
    %{imdb_id: "tt3032476", title: "Better Call Saul"}
]
```

## 7.3 Carga de dados

Para realizar a carga de dados, alguns *scripts* foram criados tanto para realizar a captura de dados quanto a inserção nas bases de dados. A captura de dados ocorreu em parte na OMDB API (maior parte dos dados de filmes e séries), no site do IMDB (classificação indicativa, receita, atores e personagens), no Wikipedia (companhias produtoras e spin-offs de séries) e em pesquisas pelo Google (companhias produtoras, spin-offs de séries e informações ausentes de algum filme/série).

Para automatizar parte do processo da alimentação das bases de dados, códigos em *javascript* foram desenvolvidos, de modo a retornar os dados nos formatos adequados para serem inseridos no banco. Tais scripts se encontram em anexo (gradi/data/scripts). Neste trabalho, tais scripts foram reformulados em Node JS e utilizados pelo Crawler (veja [Capítulo 5](#)).

Para carregar os dados existentes já capturados, basta executar os seguintes comandos na raiz do projeto:

```
1 # Registros MySQL
2 docker-compose run api mix ecto.reset
3 # Registros MongoDB
4 docker-compose run api mix run priv/repo/seeds_mongo.exs
```

## 8 Validações e inserções nas bases de dados

Esta seção trata das validações e inserções nas bases de dados dos arquivos recebidos pela aplicação.

Após a aplicação ter recebido o arquivo de um determinado cliente, este arquivo passa por uma validação. Durante esta etapa, é verificado se este arquivo está de acordo com algum dos quatro esquemas XML definidos (veja [Capítulo 6](#)), como pode-se observar no trecho de código [Listing 8.1](#) abaixo. Caso ele seja válido para algum deles, este arquivo está apto a passar para a segunda etapa. Caso contrário, ele é descartado.

```
# --- Parte omitida --- #
def schemas(:movie), do: [:movie, :moviebr]
def schemas(:series), do: [:series, :seriesbr]
# --- Parte omitida --- #
# Atraves de um arquivo XML de fato
def validate(file, type) when is_tuple(file) do
  validations = schemas(type) |> Enum.map(fn t ->
    evaluate_validation(file, t) end)
  Enum.find validations, {:error, validations}, fn {r, _} -
    -> r == :ok end
end

# Recebe um esquema e realiza a validacao do arquivo
defp evaluate_validation(file, type) do
  with {:ok, schema} <- load_schema(type) do
    case :xmerl_xsdl.validate(file, schema) do
      {:error, err} -> {:error, err}
      {new_file, _} -> {:ok, new_file}
    end
  else
    {:error, err} -> {:error, err}
    _ -> {:error, :file_load_error}
  end
end
```

Listing 8.1 – Trecho de código correspondente à validação dos arquivos para com os esquemas definidos

Uma vez validado, as séries ou filmes definidos no arquivo têm seus dados extraídos

do arquivo XML. Em seguida, estes dados são inseridos nas respectivas bases de dados (Figura 4).

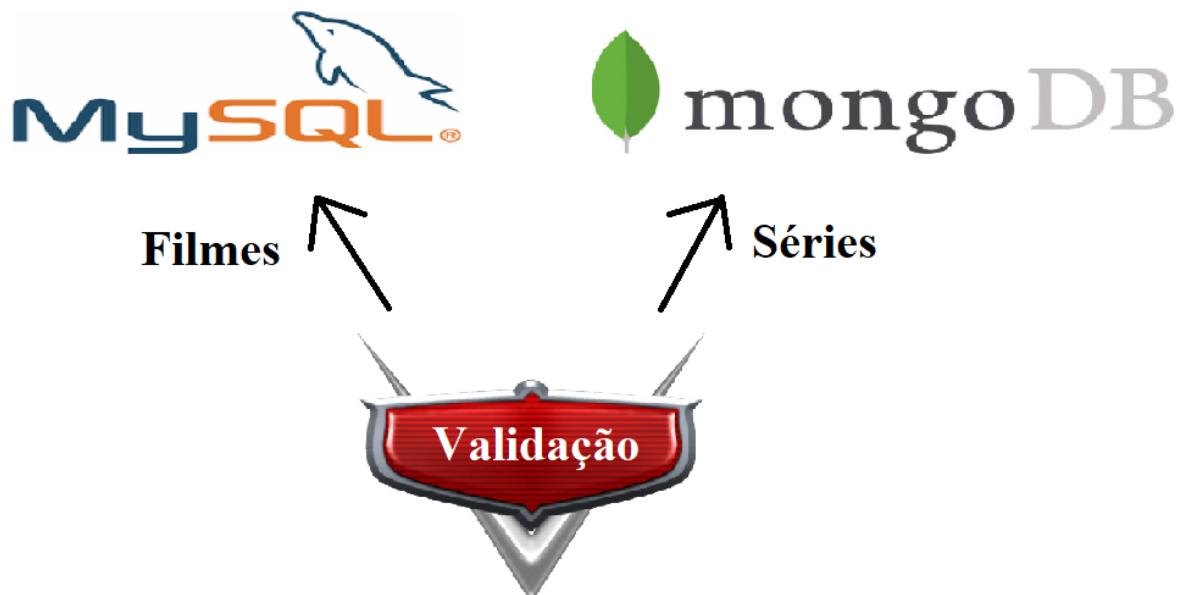


Figura 4 – Inserção após validação.

Uma das maiores dificuldade encontradas foi a inserção de filmes através de um XML. Isto se deu pelo fato de que o banco de dados para os filmes é relacional. Assim, por conta de sua granularidade, o XML tem que ser dividido nas entidades do banco e inseridos consultando os *ids* de maneira manual, enquanto nas séries o mapeamento é feito de forma praticamente automática.

Os trechos de código Listing 8.2 (referente às séries) e Listing 8.3 (referente aos filmes) abaixo descrevem os mapeamentos realizados para a inserção nas bases de dados correspondentes (não-relacional e relacional). Cada atributo é relacionado a um item do documento inserido. O XML é transformado em um mapa através de XPath, onde cada  $\sim x$  é um XPath na "raiz local". Note que o mapeamento para séries é mais direto, enquanto que para filmes há a separação de suas entidades.

```
def map_series(doc) do
  doc |> xmap(
    series: [
      ~x"//seriesset/series"1,
      imdb_id: ~x"./imdbid/text()",
```

```

    release_date: ~x"./release_date/text()",  

    # --- Parte omitida (runtime, classification, etc.)  

    actors: [  

        ~x"./actors/actor"l,  

        name: ~x"./text()",  

        character: ~x"./@character",  

        protagonist: ~x"./@protagonist" |>  

            transform_to_boolean  

    ],  

    languages: ~x"./languages/language/text()"l,  

    countries: ~x"./countries/country/text()"l,  

    writers: ~x"./writers/writer/text()"l,  

    genres: ~x"./genres/genre/text()"l
]
)
end

```

Listing 8.2 – Trecho de código que representa o mapeamento dos documentos XML (séries) para as inserções na base de dados MongoDB

```

def map_movies(doc) do
  doc |> xmap(
    movies: [
      ~x"//movies/movie"l,
      imdb_id: ~x"./imdbid/text()",  

      title: ~x"./title/text()",  

      # Parte omitida (classification, description, etc)
      characters: ~x"./characters/character/text()"l
    ],
    languages: [
      ~x"//movies/movie/languages/language"l,
      name: ~x"./text()"
    ],
    # --- Parte omitida (genres e directors)
    companies: [
      ~x"//movies/movie/companies/company"l,
      name: ~x"./text()",  

      country: ~x"./@country",
      logo: ~x"./@logo"
    ],
  )
end

```

```
    characters: [
        ~x"//movies/movie/characters/character"1,
        character: ~x"./text()", 
        actor: ~x"./@actor",
        protagonist: ~x"./@protagonist" |>
            transform_to_boolean
    ]
)
end
```

Listing 8.3 – Trecho de código que representa o mapeamento dos documentos XML (filmes) para as inserções na base de dados MySQL

## 9 Estratégias de recuperação

Nessa seção iremos discutir em alto nível quais foram as estratégias utilizadas para a recuperação dos dados do sistema, bem como uma análise em alto nível do processo.

Optamos por uma busca que prioriza precisão e não revocação, já que se trata de um sistema RI Vertical com dados que sabemos como estão armazenados. Logo, não faz muito sentido a estratégia revocação + ordenação que o google usa, já que o contexto é bem pequeno e previamente conhecido.

Para ambos os contextos, foi utilizado a seguinte estratégia:

- Remove as stopwords e usa um marcador. Ex.: "Movie of Lightning McQueen"
- "Movie | Lightning McQueen"
- Divide pelos marcadores, gerando duas strings:
- "Movie" e "Lightning McQueen"

O trecho de código [Listing 9.1](#) mostra como foi implementado essa ideia.

```
title = " " <> title <> " "
tokenized_words = [
    " i ", " me ", " my ", " myself ", " we ", " our ", " ours
    ", " ourselves ", " you ", " your ", " yours ", "
    yourself " #... (other stop words)
];
# remove as stopwords
dictionary = String.replace(
    String.downcase(title), tokenized_words, "|"
)
dictionary = String.trim(dictionary)
splitedTerms = String.split(dictionary, "|")
```

Listing 9.1 – Estratégia de separação por stopwords

## 9.1 Séries

Após o passo anterior ter sido executado, no caso das series a execução é feita em cima do banco não relacional MongoDB, logo, vamos discutir como a busca foi realizada. No caso dos filmes pesquisamos nos campos title, actors.name, actors.character, genres, languages e description.

A busca é feita através de uma regex gerada pelo BSON, usando a estratégia de \$or, pelos termos possíveis. O trecho de código Listing 9.2 mostra como isso foi implementado.

```
#gera a regex de busca do mongo para cada termo
searchTerms =
  Enum.map(splitedTerms,
    fn term -> %BSON.Regex{pattern: "#{term}", options: "i"}
      end
  )

#busca com or por termos possíveis
Map.merge(query, %{
  "$or" => [
    %{ title: %{ "$in" => searchTerms } },
    %{ "actors.name": %{ "$in" => searchTerms } },
    %{ "actors.character": %{ "$in" => searchTerms } },
    %{ genres: %{ "$in" => searchTerms } },
    %{ languages: %{ "$in" => searchTerms } },
    %{ description: %{ "$in" => searchTerms } }
  ]
})
```

Listing 9.2 – Estratégia de busca por séries

## 9.2 Filmes

Após o passo anterior ter sido executado, no caso dos filmes a execução é feita em cima do banco relacional MySQL, logo, vamos discutir como a busca foi realizada.

A partir das strings geradas, realiza-se uma busca estilo LIKE %termo% para cada um dos campos onde se há possibilidade de uma busca do usuário. No caso dos filmes pesquisamos nos campos description e title. O trecho de código Listing 9.3 mostra como isso foi implementado.

```
#gera a regex de busca do mysql para cada termo
searchTerms =
    Enum.map(splitedTerms,
        fn term -> [
            title: "%#{term}%",
            description: "%#{term}%" ]
    end
)

searchTerms = List.flatten(searchTerms)

#reduz a query em or_where para busca
Enum.reduce(searchTerms, query, fn {key, value}, q ->
    from m in q, or_where: like(field(m, ^key), ^value)
end)
```

Listing 9.3 – Estratégia de busca por filmes

## 10 Execução do sistema

Esta seção tem como objetivo demonstrar o sistema criado em execução. Assim, a tela inicial pode ser vista na [Figura 5](#).



Figura 5 – Tela inicial do sistema.

Basicamente, é possível realizar quatro operações: buscar um filme ou uma série, ou adicionar um filme ou uma série. A adição de filmes e séries é feita através de arquivos XML, conforme mostra a [Figura 6](#).

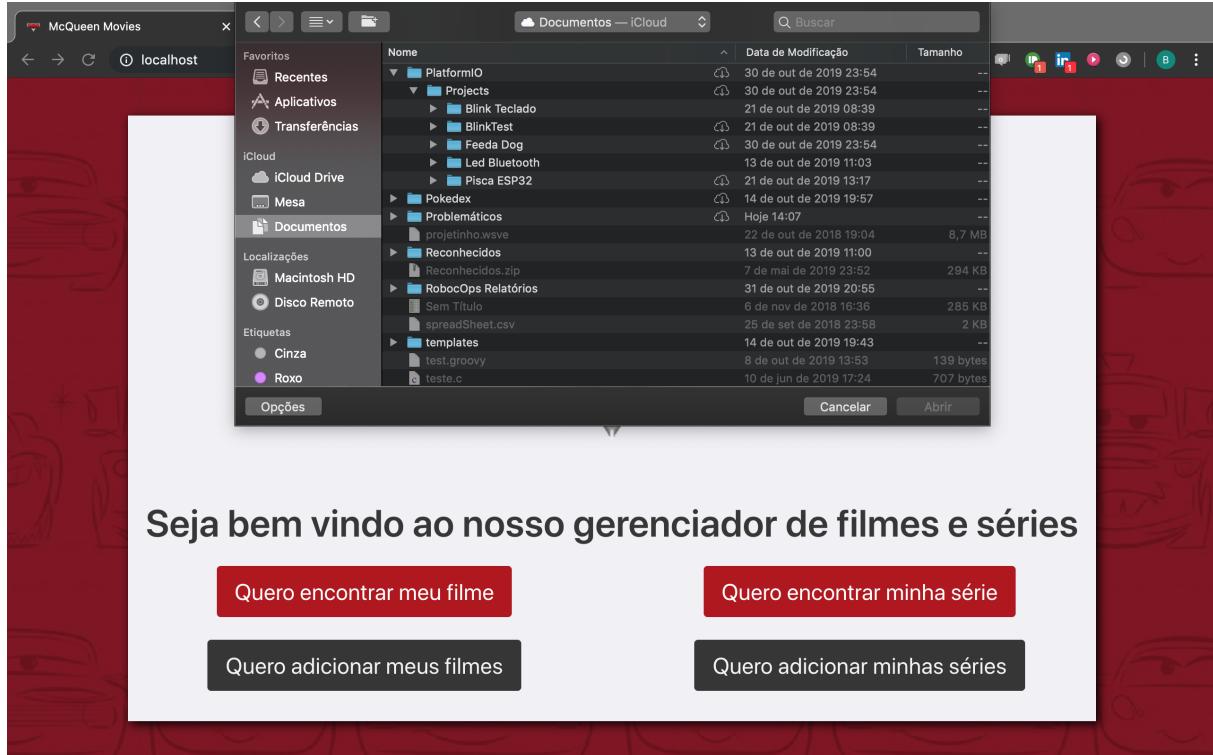


Figura 6 – Tela de adição de séries/filmes.

Nessa seção, é abordado apenas a busca por séries para simplificação da documentação. A [Figura 7](#) mostra a tela de busca das séries, com uma busca inicial de séries com o ator Terry Crews.

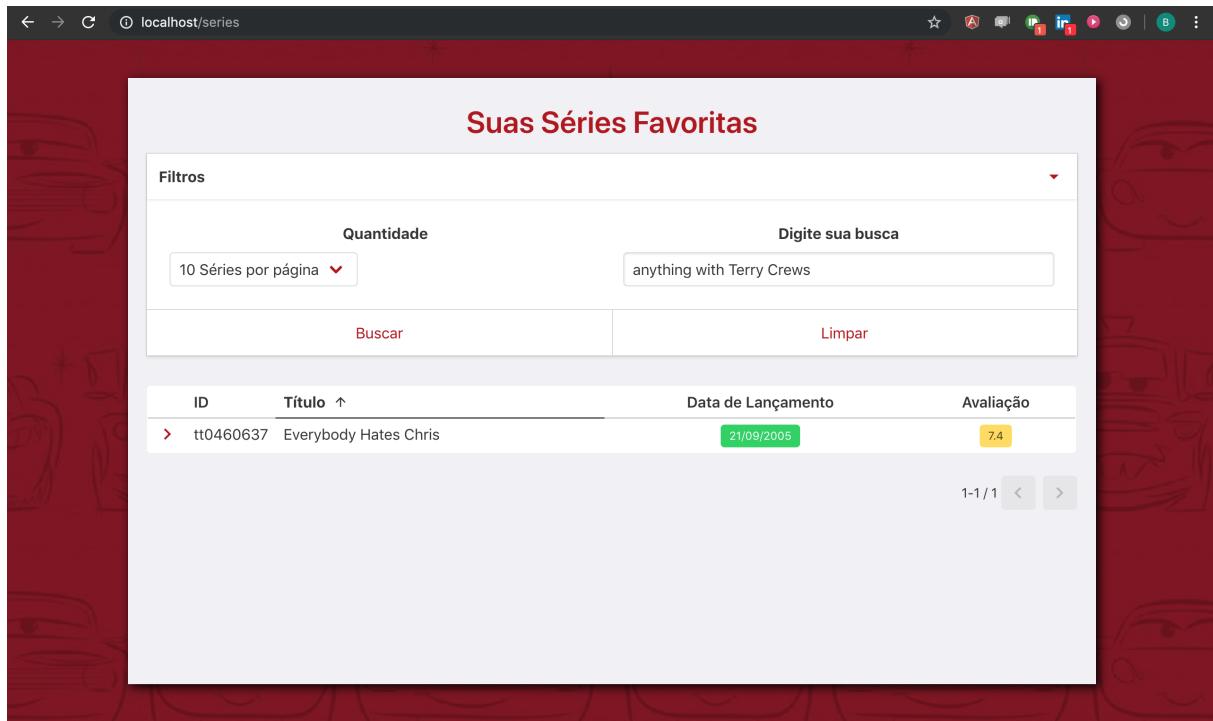


Figura 7 – Tela de busca por Terry Crews.

Após essa busca, podemos ainda fazer uma busca mais precisa, buscando ainda por séries disponíveis na língua do usuário. A Figura 8 mostra uma pesquisa feita para retornar somente filmes disponíveis em Espanhol.

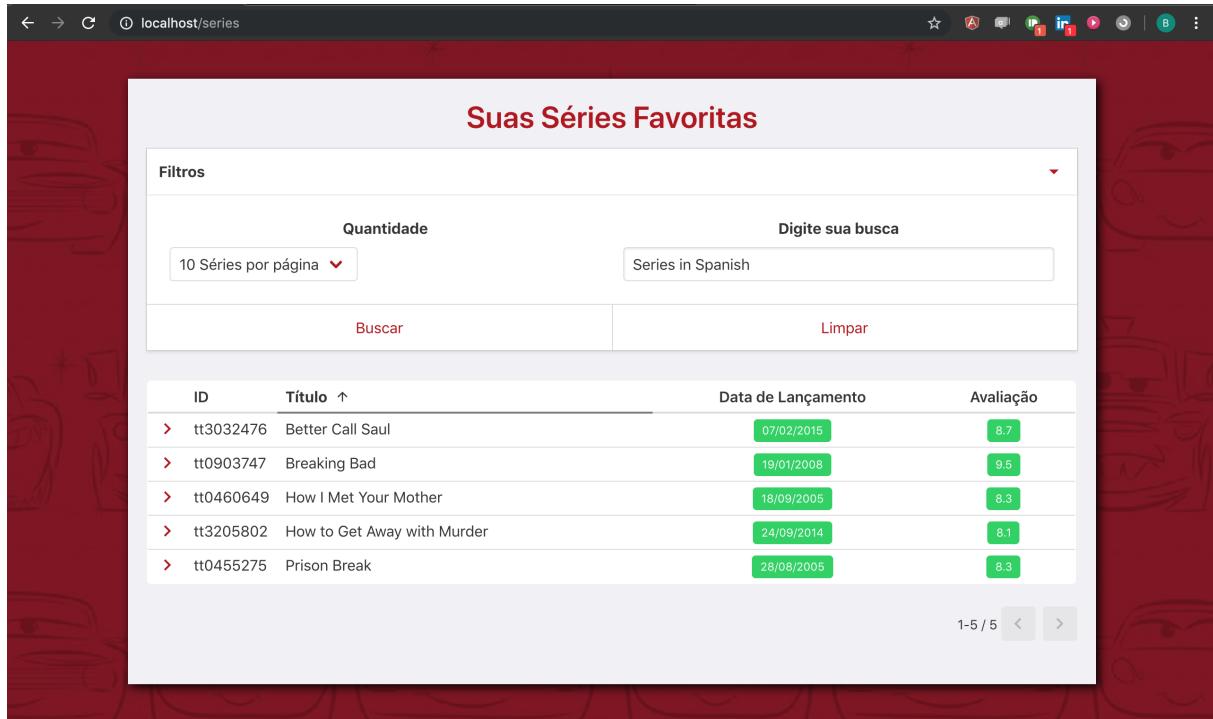


Figura 8 – Tela de busca por Séries em espanhol.

Para um exemplo de busca composta, sendo um pouco mais específico, podemos compor nossa busca por personagem OU autor, para encontrarmos a Série que desejamos. A [Figura 9](#) mostra um exemplo de busca realizada para encontrar o personagem Chris Rock ou a série Todo Mundo Odeia o Chris.

The screenshot shows a web application interface titled "Suas Séries Favoritas". At the top, there is a search bar with the placeholder text "Digite sua busca" and a text input containing "Chris Rock in Everybody Hates Chris". Below the search bar, there are two buttons: "Buscar" (Search) on the left and "Limpar" (Clear) on the right. To the left of the search bar, there is a dropdown menu labeled "Quantidade" with the option "10 Séries por página". Above the search bar, there is a section titled "Filtros" (Filters). The main area displays a table with one row of data. The table has columns: "ID", "Título ↑", "Data de Lançamento", and "Avaliação". The single row shows ID "tt0460637", Title "Everybody Hates Chris", Release Date "21/09/2005", and Rating "7.4". At the bottom of the table, there is a page navigation indicator showing "1-1 / 1" and arrows for navigating through pages.

Figura 9 – Tela de busca pela série do personagem Chris Rock OU Todo Mundo Odeia o Chris

Um outro exemplo de busca bastante corriqueira, são buscas por gênero, caso o usuário deseje assistir alguma série de um gênero específico. A [Figura 10](#) mostra um exemplo desse tipo de busca, onde o usuário procura por séries de comédia.

Suas Séries Favoritas

Filtros

Quantidade: 10 Séries por página

Digite sua busca: série of comedy

Buscar Limpar

ID	Título ↑	Data de Lançamento	Avaliação
> tt0460637	Everybody Hates Chris	21/09/2005	7.4
> tt0108778	Friends	21/09/1994	8.9
> tt0460649	How I Met Your Mother	18/09/2005	8.3
> tt0375355	Joey	08/09/2004	5.9
> tt0972412	Private Practice	18/09/2007	6.6
> tt1632701	Suits	22/06/2011	8.5
> tt0898266	The Big Bang Theory	30/04/2006	8.1

1-7 / 7 < >

Figura 10 – Tela de busca por séries de comédia

# 11 Conclusão

Neste trabalho foram realizados incrementos na plataforma de busca de filmes e séries construída no primeiro trabalho prático. Assim, como foi documentado, uma das novas funcionalidades foi a adição de filmes e séries vindos do usuário através de arquivos XML. Além disso, foi desenvolvido um *crawler* responsável por obter séries em inglês e enviá-las para a API. Tais arquivos passam por um processo de validação e posteriormente são inseridos nas bases de dados correspondentes, sendo MySQL para filmes e MongoDB para séries.

Dificuldades foram encontradas para encontrar os dados no *crawler*, principalmente pois alguns dados são obtidos via HTML, e possuem um layout complexo para realizar o processo de *parsing*. Outro ponto de dificuldade foi na inserção dos dados em um banco de dados relacional, já que cada entidade precisa ser adicionada separadamente e as relações manualmente.

Aproveitando-se do contexto definido, esquemas XML foram elaborados por meio de XSDs, permitindo que a validação dos arquivos importados pelos usuários ocorresse de forma mais facilitada. Quatro esquemas foram definidos, permitindo a criação de arquivos XML contendo tags em inglês e tags em português.

Como é um sistema de RI Vertical, notamos que a busca priorizando precisão é bastante efetiva, sendo assim podemos conseguir resultados bastante satisfatórios nesse contexto. Mesmo assim, não é exato, logo, o resultado encontrado pode não necessariamente atender às expectativas do usuário, nesse caso ele pode simplesmente tentar refazer a busca com outros termos.

# Referências

IMDB. *IMDb*. 1990. Disponível em: <<https://www.imdb.com>>. Acesso em: 02/11/2019. Citado na página 9.

JOYENT INC. *Node JS*. 2009. Disponível em: <<https://nodejs.org/en/about/>>. Acesso em: 02/11/2019. Citado na página 9.

OMDBAPI. *The Open Movie Database*. 2014. Disponível em: <<http://www.omdbapi.com/>>. Acesso em: 02/11/2019. Citado na página 9.

W3C. *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. 2001. Disponível em: <<https://www.w3.org/TR/xmlschema11-1>>. Acesso em: 20/10/2019. Citado na página 17.

W3C. *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. 2001. Disponível em: <<https://www.w3.org/TR/xmlschema11-2>>. Acesso em: 20/10/2019. Citado na página 17.

W3C. *XML Schema*. 2001. Disponível em: <<https://www.w3.org/2001/XMLSchema>>. Acesso em: 20/10/2019. Citado na página 17.