



UNIVERSIDADE DO MINHO

MESTRADO EM ENGENHARIA INFORMÁTICA

PROCESSAMENTO E REPRESENTAÇÃO DE INFORMAÇÃO  
&  
GRAMÁTICAS NA COMPREENSÃO DE SOFTWARE

---

**iBanda**

---

Diana Ribeiro Barbosa - A78679  
José Carlos Lima Martins - A78821

1 de Fevereiro de 2019

## Resumo

O projeto prático sobre o qual este relatório incide tem como nome *iBanda - Arquivo Digital Musical* e consiste numa aplicação *web* que implementa um repositório digital de obras musicais e respetivas partituras, respeitando a estrutura do modelo de referência internacional OAIS ("Open Archive Information System"). O projeto destina-se a ser usado por uma banda filarmónica para aceder às partituras, bastando apenas terem acesso à *internet* e uma conta válida.

O sistema permite ainda a gestão de uma agenda de eventos, gestão de notícias, gestão dos utilizadores/músicos da banda, gestão de repertórios e gestão de uma pequena biblioteca de suporte.

No primeiro capítulo, fazemos uma breve introdução ao tema. No capítulo 2 detalhamos os *schemas* desenvolvidos e decisões tomadas relativamente à base de dados, no capítulo 3 explicamos algumas das funções criadas na componente dos *controllers*, no capítulo 4 incidimos sobre os vários *routes*, no capítulo 5 falamos sobre a nossa abordagem tomada relativamente às *views*, no capítulo 6 identificamos as várias localizações dos ficheiros que são armazenados, no capítulo 7 detalhamos o processo de criação das gramáticas criadas, no capítulo 8 explicamos como fazer a instalação do sistema e, por fim, no capítulo 9 mencionamos algumas conclusões a que chegamos após a realização deste trabalho.

## Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Models</b>	<b>3</b>
<b>3</b>	<b>Controllers</b>	<b>4</b>
<b>4</b>	<b>Routes</b>	<b>6</b>
4.1	API . . . . .	6
4.2	Routers visíveis externamente . . . . .	8
4.2.1	Ingestão . . . . .	12
4.2.2	Administração . . . . .	14
4.2.3	Disseminação . . . . .	14
4.2.4	Biblioteca de suporte . . . . .	14
4.3	Autenticação e Permissões . . . . .	14
<b>5</b>	<b>Views</b>	<b>15</b>
<b>6</b>	<b>Localização dos ficheiros armazenados</b>	<b>17</b>
<b>7</b>	<b>Gramáticas</b>	<b>17</b>
7.1	Agenda . . . . .	17
7.2	News . . . . .	20
7.3	Interligação entre Gramáticas e Web site . . . . .	22
<b>8</b>	<b>Instalação</b>	<b>24</b>
<b>9</b>	<b>Conclusões</b>	<b>25</b>
<b>A</b>	<b>Gramática da Agenda</b>	<b>26</b>
<b>B</b>	<b>Gramática das News</b>	<b>28</b>

# 1 Introdução

Este trabalho prático consiste numa aplicação web que implementa um repositório digital de obras musicais e respetivas partituras, respeitando a estrutura do modelo de referência internacional **OAIS** (*"Open Archive Information System"*).

Neste modelo existem 3 tipos de atores: o utilizador do tipo **produtor** que deposita partituras e cataloga as obras, o **administrador** que gere e mantém o sistema e, por fim, o **consumidor** que consulta e descarrega a informação disponibilizada. Assim, segue que cada um pertencerá a um de três tipos de utilizadores do sistema, com permissões específicas à sua posição segundo o modelo *OAIS*.

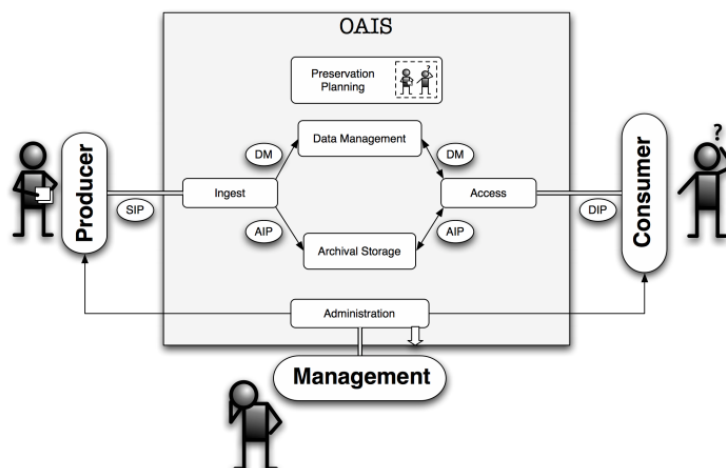


Figura 1: Estrutura base de um repositório do modelo *OAIS*.

Relativamente à estrutura da aplicação, segue o modelo **MVC** (*Model-View-Controller*), sendo que a base de dados utilizada foi o *MongoDB*.

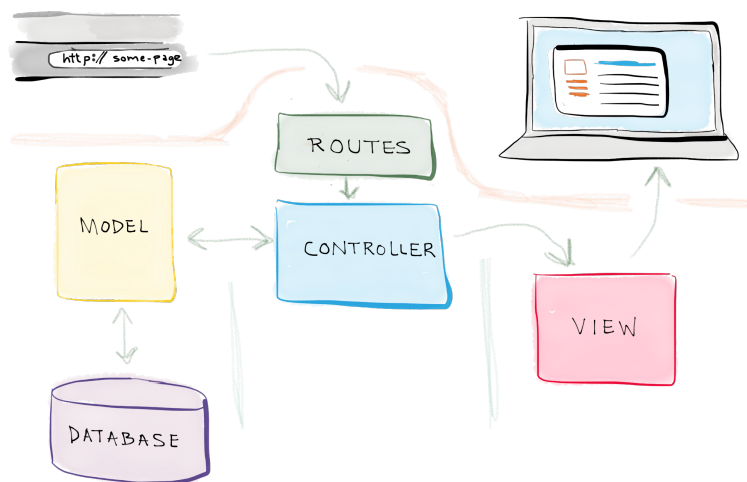


Figura 2: Estrutura base da aplicação

## 2 Models

É nos **Models** que se encontra definido o *schema* da base de dados da aplicação, nomeadamente, as coleções de documentos em MongoDB (BD não relacional). Cada ficheiro `.js` corresponde a uma coleção, sendo que cada **schema** permite definir a estrutura de cada tipo de documento.

```
var mongoose = require("mongoose")
var Schema = mongoose.Schema

var EntrySchema = new Schema(
  {
    desc: {type: String, required: true}
  }
)

module.exports = mongoose.model("Entry", EntrySchema, "entries")
```

*Ficheiro `entry.js` da pasta `Models`.*

É de notar que utilizamos os identificadores criados pelo MongoDB (`_id`), em vez de criar um atributo `id` para cada tipo de documento, sendo os seguintes os documentos definidos:

**Article:** define um artigo noticiário e apresenta a seguinte estrutura:

- **title:** String obrigatória que define o título do artigo;
- **subtitle:** String que define o subtítulo do artigo;
- **date:** String obrigatória que define a data;
- **authors:** Lista de Strings, cada uma com o nome de um autor do artigo;
- **body:** String obrigatória que representa o corpo do artigo;
- **topics:** Lista de Strings, cada uma com um tópico;
- **visible:** Valor booleano obrigatório que permite definir se o artigo deve estar visível ou não para os produtores e para os consumidores.

Na inserção de artigos é verificado se a data é igual ou antecedente ao dia corrente.

**Entry:** identifica cada entrada na biblioteca de suporte, apresentando a seguinte estrutura:

- **desc:** String obrigatória que possui a descrição do documento PDF.

**Event:** identifica cada evento de uma agenda. Apresenta a seguinte estrutura:

- **title:** String obrigatória que representa o título do evento;
- **desc:** String com a descrição do evento;
- **local:** String com o local do evento;
- **startDate:** String obrigatória que possui a data inicial do evento (yyyy-mm-dd);
- **startHour:** String obrigatória com a hora inicial do evento (hh:mm);
- **endDate:** String obrigatória com a data final do evento (yyyy-mm-dd);
- **endHour:** String obrigatória com a hora final do evento (hh:mm).

Na inserção de eventos é verificado se a data+hora inicial antecede a data+hora final.

**Piece:** representa parte do AIP de uma obra. É aqui que são guardados todos os meta-dados de uma obra. Apresenta a seguinte estrutura:

- **title:** String obrigatória que representa o título da obra;
- **composer:** String com o compositor da obra;
- **type:** String obrigatória que define o tipo da obra;
- **arrangement:** String com quem fez o arranjo da obra;
- **instruments:** Obrigatório, possui os instrumentos. Cada instrumento possui:
  - **name:** String obrigatória com o nome do instrumento;
  - **score:** Obrigatória, possui informação sobre a partitura;
    - \* **voice:** String com a voz;
    - \* **clave:** String com a clave;
    - \* **tune:** String com a afinação;
    - \* **path:** String obrigatória com o caminho da partitura no ficheiro zip.

**User:** representa cada utilizador do sistema. Apresenta a seguinte estrutura:

- **name:** String obrigatória com o nome do utilizador;
- **email:** String obrigatória com o email;
- **password:** String obrigatória com a password;
- **type:** String obrigatória que define o tipo de utilizador, ou seja, define o nível de privilégios que o utilizador tem. Tipo 1 (**Administrador**) tem quase todos os privilégios. Tipo 2 (**Produtor**), pode introduzir obras, pode apagar(DELETE's)/editar(PUT's)/inserir(POST's)/ver(GET's) instrumentos e tem acesso(GETs) às obras, agenda, notícias e biblioteca de suporte. Tipo 3 (**Consumidor**) apenas tem acesso(GETs) às obras, agenda, notícias e biblioteca de suporte;
- **approved:** Boolean obrigatório que permite saber se a conta já foi aprovada pelo administrador
- **stats:** lista que possui as estatísticas do utilizador;
  - **idPiece:** id (String) da obra a que se associa os valores seguintes;
  - **views:** número de visualizações do utilizador dessa obra;
  - **downloads:** número de downloads do utilizador dessa obra.

### 3 Controllers

Os **controllers** possuem várias funções que permitem manusear os documentos e seus valores. Tal como mencionado anteriormente, as coleções e respetivos documentos são estruturados por **Schemas** recorrendo ao **Mongoose**. E é aqui, nos **Controllers**, que se encontram definidos a criação, atualização, destruição, listagem e obtenção de estatísticas dos mesmos.

```
var Article = require("../models/article")
const Articles = module.exports

Articles.list = () => {
  return Article
    .find()
    .sort({date: -1})
}
```

```

        .exec()
    }

    Articles.listVisibles = () => {
        return Article
            .find({visible: true})
            .sort({date: -1})
            .exec()
    }

    Articles.getArticle = id => {
        return Article
            .findOne({_id: id})
            .exec()
    }

```

*Excerto do ficheiro `article.js` da pasta `controllers`.*

**Article:** permite a criação (`createArticle`, `insertMany`), atualização (`updateArticle`) e destruição (`deleteArticle`) de artigos. Para além disso, permite listar todos os artigos (`list`) ou apenas aqueles que estão visíveis (`listVisibles`). Permite também obter um único artigo dado o seu id (`getArticle`), bem como mudar a visibilidade de um artigo dado um id (`changeVisibility`). Por fim, permite a obtenção de artigos consoante a data (`getArticlesByDate`), o autor (`getArticlesByAuthor`) e o tópico (`getArticlesByTopic`).

**Entry:** permite a criação (`createEntry`), atualização (`updateEntry`) e destruição (`deleteEntry`) de entradas na biblioteca de suporte. Para além disso permite listar todas as entradas (`list`) ou obter uma única entrada dado o seu id (`getEntry`).

**Event:** permite a criação (`createEvent`, `insertMany`), atualização (`updateEvent`) e destruição (`deleteEvent`) de eventos. Para além disso, permite listar todos os eventos (`list`) ou obter um único evento dado o seu id (`getEvent`). Por fim, permite a obtenção de eventos consoante o evento se realize numa dada data (`getEventsByDate`), numa dada hora (`getEventsByHour`), numa dada data e hora (`getEventsByDataHour`) ou num dado local (`getEventsByLocal`).

**Piece:** permite a criação (`createPiece`), atualização (`updatePiece`) e destruição (`deletePiece`) na parte da obra armazenada na base de dados. Para além disso, permite listar todas as obras (`list`) ou obter uma obra dado o seu id (`getPiece`), e de novo apenas a parte armazenada na base de dados. Para além disso, o `addInst` permite adicionar um instrumento a uma obra, o `remInst` permite eliminar um instrumento da obra e o `updateInst` permite atualizar um instrumento de uma obra.

**User:** permite a criação (`createUser`), atualização (`updateUser`) e destruição (`deleteUser`) de utilizadores. Para além disso permite listar todos os utilizadores (`list`). Permite também obter um único utilizador dado o seu id (`getUser`) ou dado o seu email (`findOne`). Na criação de utilizadores a password é encriptada sendo que existe também uma função por forma a verificar se uma password não encriptada é igual à armazenada (`isValidPassword`), função importante para o login de utilizadores. Existem também três funções que permitem manter as estatísticas atualizadas, `updateViews` para adicionar uma visualização de um artigo, `updateDownloads` para adicionar um download de um artigo e `deleteStats` para apagar uma entrada das estatísticas quando a obra é apagada do sistema. Há também o `approve` que recebe o id da conta e aprova-a, bem como, foi criada a função `updatePassword` que permite que um utilizador atualize a sua password, onde é necessário providenciar a antiga password e a nova password.

Por fim, há um conjunto de funções que permite a obtenção das estatísticas: `getViews` que obtém o número de visualizações total para um utilizador, `getDownloads` que obtém o número de downloads total para um utilizador, `getPiece` que obtém a obra com mais visualizações por um utilizador, `getdownload`

que obtém a obra com mais downloads por um utilizador, `getAllViews` que devolve o número de visualizações total, `getAllDownloads` que devolve o número de downloads total, `getUsersViews` que obtém uma lista com as visualizações totais de cada utilizador, `getUsersDownloads` que obtém uma lista com os downloads totais de cada utilizador, `getUserWithMostViews` que obtém o utilizador com mais visualizações, `getUserWithMostDownloads` que obtém o utilizador com mais downloads, `getMostViewed` que devolve a obra com mais visualizações e `getMostDownloads` que devolve a obra com mais downloads.

## 4 Routes

### 4.1 API

A API permite que pedidos do cliente possam realizar a chamada de funções (necessárias à sua execução) definidas nos `Controllers`, desde que este esteja autenticado e possua permissões para tal chamada, consoante o URL e o método do pedido. De seguida, apresentam-se as rotas e a função que é chamada para cada rota.

**Article:** `/api/article` concatenado com:

- método GET
  - `/date/:date` - `getArticlesByDate`
  - `/author/:author` - `getArticlesByAuthor`
  - `/topic/:topic` - `getArticlesByTopic`
  - `/:id` - `getArticle`
  - `/` - `list`
- método POST
  - `/insertMany` - `insertMany`
  - `/` - `createArticle`
- método PUT
  - `/visible/:id` - `changeVisibility`
  - `/:id` - `updateArticle`
- método DELETE
  - `/:id` - `deleteArticle`

**Entry:** `/api/entry` concatenado com:

- método GET
  - `/:id` - `getEntry`
  - `/` - `list`
- método POST
  - `/` - `createEntry`
- método PUT
  - `/:id` - `updateEntry`
- método DELETE
  - `/:id` - `deleteEntry`

**Event: /api/event concatenado com:**

- método GET
  - /date/:date - getEventsByDate
  - /hour/:hour - getEventsByHour
  - /date\_hour?date=:date&hour=:hour - getEventsByDateHour
  - /local/:local - getEventsByLocal
  - /:id - getEvent
  - / - list
- método POST
  - /insertMany - insertMany
  - / - createEvent
- método PUT
  - /:id - updateEvent
- método DELETE
  - /:id - deleteEvent

**Piece: /api/piece concatenado com:**

- método GET
  - /:id - getPiece
  - / - list
- método POST
  - /addInst/:id - addInst
  - / - createPiece
- método PUT
  - /updInst?idP=idPiece&idI=idInst - updateInst
  - /:id - updatePiece
- método DELETE
  - /remInst?idP=idPiece&idI=idInst - remInst
  - /:id - deletePiece

**Statistic: /api/statistic concatenado com:**

- método GET
  - /views/most/:id - getPieceViews
  - /downloads/most/:id - getPieceDownloads
  - /views/:id - getViews
  - /downloads/:id - getDownloads
  - /viewsAll - getAllViews
  - /downloadsAll - getAllDownloads



- /viewsMostUser - getUserWithMostViews
- /downloadsMostUser - getUserWithMostDownloads
- /viewsMost - getMostViewed
- /downloadsMost - getMostDownloaded
- /usersViews - getUsersViews
- /usersDownloads - getUsersDownloads

**User: /api/user concatenado com:**

- método GET
  - /approve/:id - approve
  - /:id - getUser
  - / - list
- método POST
  - / - createUser
- método PUT
  - /views/:id - updateViews
  - /downloads/:id - updateDownloads
  - /deleteStat - deleteStat
  - /updPass/:id - updatePassword
  - /:id - updateUser
- método DELETE
  - /:id - deleteUser

## 4.2 Routers visíveis externamente

Na maioria dos casos as rotas fazem um pedido à API e com os dados obtidos renderizam uma vista. Contudo, há outras rotas que realizam outras tarefas, bem como diversos pedidos à API. Abaixo, exemplificamos as rotas acessíveis e, de forma sucinta, as suas funções:

**/articles concatenado com:**

- Método GET:
  - /authors/:author - Obtém os artigos com autor “author” e apresenta-os numa lista
  - /topics/:topic - Obtém os artigos com o tópico “topic” e apresenta-os numa lista
  - /article/:id - Obtém o artigo e apresenta um formulário com os valores armazenados de forma a o utilizador editar
  - /list/:date - Obtém os artigos com a data igual a “date” e apresenta-os (neste caso apenas é gerado os li’s sendo depois concatenado com o html via jquery)
  - /list - Obtém todos os artigos e apresenta-os (neste caso apenas é gerado os li’s sendo depois concatenado com o html via jquery)
  - /grammar - Apresenta ao utilizador um formulário de forma a poder escrever texto que respeite a gramática definida, apresentando os erros quando o mesmo não respeita, e quando o mesmo respeitar pedir a inserção na base de dados dos artigos presentes no texto

- /article - Apresenta ao utilizador o formulário para a inserção de um artigo
- /:id - Obtém o artigo com “id” e apresenta a sua informação ao utilizador
- / - Apresenta todos os artigos/menu principal dos artigos ao utilizador
- Método POST:
  - /grammar - É para aqui enviado o texto referente ao formulário da gramática, sendo aqui feito o parse do mesmo e em caso de erro volta à página do formulário, em caso de sucesso pede a inserção dos artigos na base de dados e em caso de sucesso o utilizador é redirecionado para /articles
  - / - Recebe os valores referentes ao formulário da inserção de um artigo e em caso de sucesso redireciona para /articles
- Método PUT:
  - /visible/:id - Recebe o pedido de tornar visível ou invisível um artigo, em caso de sucesso redireciona para o artigo alterado
  - /:id - Recebe os valores do formulário de edição de um artigo, faz o pedido de alteração à API e em caso de sucesso redireciona para o artigo alterado

#### **/entries concatenado com:**

- Método GET:
  - /entry/:id - Obtém a entrada e apresenta um formulário com os valores armazenados de forma a o utilizador editar
  - /entry - Apresenta ao utilizador o formulário para a inserção de uma entrada na biblioteca de suporte
  - /:id - Obtém a entrada com “id” e apresenta a sua informação ao utilizador
  - / - Apresenta todas as entradas/menu principal das entradas ao utilizador
- Método POST:
  - / - Recebe os valores referentes ao formulário da inserção de uma entrada, verifica se o ficheiro é um pdf, em caso afirmativo pede a inserção de informação na base de dados e se houver sucesso move o ficheiro para /public/pdfs/ e em caso de sucesso redireciona para /entries
- Método PUT:
  - /:id - Recebe os valores referentes ao formulário de edição de uma entrada. Caso seja inserido um ficheiro, é atualizado esse ficheiro (caso seja um pdf) bem como a informação alterada. Caso não seja inserido um ficheiro, apenas é alterada a informação presente na base de dados
- Método DELETE:
  - /:id - Pede a eliminação da entrada na base de dados e em caso de sucesso, elimina o ficheiro armazenado, redirecionando para /entries

#### **/events concatenado com:**

- Método GET:
  - /event/:id - Obtém o evento e apresenta um formulário com os valores armazenados de forma a o utilizador editar
  - /list/:date - Obtém os eventos que se realizam/estão a ser realizados em “date” e apresenta-os (neste caso apenas é gerado os li’s sendo depois concatenado com o html via jquery)

- /list - Obtém todos os eventos e apresenta-os (neste caso apenas é gerado os li's sendo depois concatenado com o html via jquery)
  - /grammar - Apresenta ao utilizador um formulário de forma a poder escrever texto que respeite a gramática definida, apresentado os erros quando o mesmo não respeita, e quando o mesmo respeitar pedir a inserção na base de dados dos eventos presentes no texto
  - /event - Apresenta ao utilizador o formulário para a inserção de um evento
  - /export - Exportação de todos os eventos num ficheiro em formato json
  - /:id - Obtém o evento com "id" e apresenta a sua informação ao utilizador
  - / - Apresenta todos os eventos/menu principal dos eventos ao utilizador
- Método POST:
    - /grammar - É para aqui enviado o texto referente ao formulário da gramática, sendo aqui feito o parse do mesmo e em caso de erro volta à página do formulário, em caso de sucesso pede a inserção dos eventos na base de dados e em caso de sucesso o utilizador é redirecionado para /events
    - / - Recebe os valores referentes ao formulário da inserção de um evento e em caso de sucesso redireciona para /events
  - Método PUT:
    - /:id - Recebe os valores do formulário de edição de um evento, faz o pedido de alteração à API e em caso de sucesso redireciona para o evento alterado
  - Método DELETE:
    - /:id - Pede a eliminação do evento na base de dados e em caso de sucesso redireciona para /events

#### **/pieces concatenado com:**

- Método GET:
  - /ingestion - apresenta o formulário para a inserção do ficheiro zip para ingestão do mesmo
  - /addInst/:id - apresenta o formulário para a inserção de um instrumento para a obra "id"
  - /updInst?idP=idPiece&idI=idInst - Obtém os valores do instrumento "idInst" e apresenta-os num formulário de forma a o utilizador editar (deve ser usado para atualizar o ficheiro associado a um instrumento)
  - /export/:id - realiza a exportação da obra com id "id", gera o iBanda-SIP.json obtendo o piece da base de dados e apagando os \_id's presentes, cria o zip com este ficheiro e os ficheiros armazenados desta obra e por fim envia o zip ao utilizador
  - /piece/:id - Obtém a obra e apresenta um formulário com os valores armazenados de forma a o utilizador editar
  - /:id - Obtém a obra com "id" e apresenta a sua informação ao utilizador
  - / - Apresenta todas as obras/menu principal das obras ao utilizador
- Método POST:
  - /addInst/:id - recebe os valores do formulário de adição de um novo instrumento, verificando se o ficheiro é um pdf, verifica se não existe na obra outro ficheiro com o mesmo nome, se sim usa o id do instrumento como nome, guarda o ficheiro e por fim pede a inserção do instrumento na base de dados
  - / - É aqui ingerido o ficheiro zip, verificado se é mesmo um ficheiro zip, verificado se o manifesto existe, se os ficheiros que referencia estão no zip, pede a inserção da informação presente no manifesto na base de dados e em caso de sucesso extrai os ficheiros referenciados para o sistema

- Método PUT:

- /updInst?idP=idPiece&idI=idInst - Recebe os valores do formulário de edição de um instrumento, verifica se o ficheiro é um pdf, elimina o ficheiro que estava associado ao instrumento, verifica se não existe na obra outro ficheiro com o mesmo nome que o novo ficheiro, se sim usa o id do instrumento como nome, guarda o novo ficheiro, faz o pedido de alteração à API e em caso de sucesso redireciona para a obra alterada
- /:id - Recebe os valores do formulário de edição de uma obra, faz o pedido de alteração à API e em caso de sucesso redireciona para a obra alterada

- Método DELETE:

- /remInst?idP=idPiece&idI=idInst - pede a remoção do instrumento da base de dados e, em caso de sucesso, elimina o ficheiro que lhe estava associado
- /:id - Pede a eliminação da obra na base de dados, elimina os ficheiros referentes a essa obra e, em caso de sucesso redireciona para /pieces

### /statistics concatenado com:

- Método GET:

- /:id - Obtém várias estatísticas do utilizador com “id” e apresenta-as ao utilizador
- / - Obtém todos os utilizadores, bem como algumas das suas estatísticas, para além de algumas estatísticas globais ao sistema e apresenta-as ao utilizador

### /users concatenado com:

- Método GET:

- /approve/:id - Recebe o id de uma conta e tenta aprová-la, redireciona para a página com a informação da conta (/users/:id)
- /user/:id - Obtém o utilizador e apresenta um formulário com os valores armazenados de forma a o utilizador editar
- /user - Apresenta ao utilizador o formulário para se registar
- /updPass/:id - Apresenta ao utilizador o formulário para atualizar a password do utilizador
- /:id - Obtém o utilizador com “id” e apresenta a sua informação ao utilizador
- / - Apresenta todos os utilizadores/menu principal dos utilizadores ao utilizador

- Método POST:

- Recebe os valores referentes ao formulário do registo de um utilizador, caso o utilizador seja criado com sucesso, redireciona para o login (/)

- Método PUT:

- /updPass/:id - Recebe os valores do formulário de atualização de password do utilizador, verifica se é o utilizador da sessão a fazer o pedido, faz o pedido de alteração à API e em caso de sucesso redireciona para o menu principal
- /:id - Recebe os valores do formulário de edição de um utilizador, faz o pedido de alteração à API e em caso de sucesso redireciona para o user alterado, apaga todas as sessões do user com id “id” de forma a garantir que o utilizador “id” tem as permissões logo atualizadas

- Método DELETE:

- /:id - Pede a eliminação de um utilizador na base de dados e em caso de sucesso redireciona para /users, apaga as sessões do user id “id” de forma a garantir que durante o tempo até à expiração do token ele já não tenha acesso

**/(index) concatenado com:**

- Método GET:
  - /main - Área principal de um utilizador, obtém os artigos e apresenta-os ao utilizador. Permite navegar pelas várias zonas a que o utilizador tem acesso
  - /logout - Realiza o logout do utilizador, ao apagar a sua sessão(token) do sistema. Redireciona para a área de login
  - / - Apresenta um formulário para a realização do login
- Método POST:
  - /login - Recebe os valores do formulário de login, verifica se o utilizador existe, se a conta já foi aprovada pelo administrador, bem como se a password é a correta e cria por fim o token, armazenando na sessão o token bem como o seu \_\_id e permissões do mesmo, por fim, em caso de sucesso, redireciona para a sua área principal

#### 4.2.1 Ingestão

Para que o SIP (*Submission Information Package*) seja aceite pelo sistema, este deve estar de acordo com as seguintes regras:

- Ser um ficheiro em formato zip
- O manifesto (iBanda-SIP.json) tem de estar na raiz do ficheiro zip
- O nome do manifesto tem de ser iBanda-SIP.json
- Os restantes ficheiros estão ao nível do manifesto ou em pastas (estando estas ao nível do manifesto) no ficheiro zip
- O manifesto tem que respeitar a sintaxe do JSON.
- O manifesto tem que possuir os seguintes atributos:
  - title
  - type
  - composer (opcional)
  - arrangement (opcional)
  - instruments (pelo menos um instrumento):
    - \* name
    - \* score
      - voice (opcional)
      - clave (opcional)
      - tune (opcional)
      - path (não deve incluir “./” nem “~/” nem “/” no início do caminho porque, este caminho deve ser o caminho do ficheiro dentro do zip e não no sistema de ficheiros)
- Os ficheiros indicados no manifesto têm que existir no ficheiro zip. Caso não existam, o SIP não é importado para o sistema.
- Apenas são extraídos para o sistema os ficheiros definidos no manifesto.

```

{
  "title": "musica",
  "type": "concerto",
  "instruments": [
    { "name": "instrumento",
      "score": {
        "path": "score/score.txt"
      }
    }
  ]
}

```

*Exemplo de um manifesto básico (assumindo que o seu nome é `iBanda-SIP.json` e que o ficheiro `score.txt` está numa pasta “score” dentro do ficheiro zip).*

Por forma a importar o DataSet fornecido pelo professor, foi criado um script (*convertAndImport.sh*) que recebe como argumento os ficheiros a importar, converte-os para um estado suportado pelo sistema desenvolvido e, no fim, faz a importação dos mesmos para o `mongoDB`. A conversão consiste em passar os nomes dos atributos para inglês (os que são usados no sistema) e na eliminação do `_id` de forma a ser criado um pelo `mongoDB`. Contudo, como este DataSet não inclui os ficheiros associados aos caminhos(path), nestas obras no sistema não é possível, obviamente, visualizar tais ficheiros. É também, após a importação, adicionado um `_id` a cada instrumento em cada documento, de maneira que seja possível a atualização de cada instrumento e, como tal, caso os ficheiros sejam adicionados na pasta `public/scores/{_id da obra}/` e tenham os mesmos nomes que está no path da obra, então passarão a estar acessíveis para os utilizadores. Portanto, para importar, basta `./convertAndImport.sh <caminhoFicheiro1> <caminhoFicheiro2> ....` Por forma a associar os ficheiros a estes instrumentos sem ficheiros, pode também ser usado a interface de atualização de instrumentos do sistema.

```

#!/bin/bash

#Convert attribute names to english ones
sed -i '/\_id\[\"[^\n]*\/d' "$@"
sed -i "s/titulo/title/" "$@"
sed -i "s/tipo/type/" "$@"
sed -i "s/compositor/composer/" "$@"
sed -i "s/instrumentos/instruments/" "$@"
sed -i "s/arranjo/arrangement/" "$@"
sed -i "s/nome/name/g" "$@"
sed -i "s/partitura/score/g" "$@"
sed -i "s/voz/voice/g" "$@"
sed -i "s/afinacao/tune/g" "$@"

#Import files to mongoDB
for f in $*
do
  mongoimport -d iBanda -c pieces --file "$f"
done

#Create _id for instruments necessary to update them on system
mongo iBanda --eval 'db.pieces.find().forEach(function(doc) {
  for (var i = 0; i < doc.instruments.length; i++) {
    doc.instruments[i]._id = ObjectId()
  }
  db.pieces.update({ "_id" : doc._id }, doc)
})'

```

*Script `convertAndImport.sh`*

#### 4.2.2 Administração

A administração do sistema é a gestão interna de todos os tipos de objetos/documentos presentes desde obras, a eventos, a utilizadores, etc. Esta gestão compreende, a adição, edição e remoção destes documentos bem como a visualização de estatísticas referentes ao sistema.

Como já apresentado anteriormente estas funcionalidades estão presentes no sistema, nomeadamente nos routers pieces e statistics, sendo que o administrador apenas não pode inserir obras e utilizadores.

#### 4.2.3 Disseminação

Quanto à disseminação qualquer utilizador autenticado tem acesso à mesma, sendo que cada obra pode ser exportada para um ZIP ou visualizada por uma interface. O ficheiro ZIP exportado possui a mesma estrutura de um SIP, visto possuir um iBanda-SIP.json com os metadados que estavam armazenados na base de dados (sendo que são apagados os vários `_id`'s por forma a não estarem incluídos no json criado) e os ficheiros que são referenciados pela obra.

#### 4.2.4 Biblioteca de suporte

A biblioteca de suporte consiste de um conjunto de ficheiros PDF com um campo descritivo associado (por exemplo, "*25 estudos para flauta de X*"). De modo a suportar tal biblioteca, a descrição contida em cada um dos ficheiros é armazenada na base de dados (na coleção *entries*). Já os PDFs são armazenados na pasta `pdfs` pertencente à `public`. Estes, são guardados com o nome igual ao seu identificador da entrada na base de dados de modo a garantir que não há nomes repetidos e portanto, não há erros desse tipo no processo de armazenamento.

### 4.3 Autenticação e Permissões

De modo a assegurar a autenticação dos utilizadores, o sistema recorre ao package `passport` em conjunto com o JWT (`tokens`). O `admin`, caso este não exista na base de dados, será criado um ao executar o sistema (com `email root@root` e `password 'ibanda'`). Desta forma, quando um `user` efetua o *login*, é criado um `token` que lhe permite passar pelas áreas protegidas do sistema.

O token tem um tempo de expiração de 30 minutos. Na eventualidade de isso ocorrer, o utilizador necessita de reefetuar o login, o que corresponde à criação de um novo token e consequente armazenamento do mesmo na sessão do user. É ainda armazenado o seu tipo (de utilizador) nomeadamente as permissões que possui e o seu `_id` na sessão. A criação/validação do token é feita usando uma chave privada (para o criar) e uma pública (para o validar).

Por forma a restringir os utilizadores consoante o seu nível de permissões, isto é, permitir ou não o acesso sempre que este é pedido, o sistema recorre ao valor `type` do utilizador armazenado na sessão para tomar essa decisão. Este valor também permite às `views` apresentarem o conteúdo adequado consoante o tipo de utilizador. Já a presença do `_id` na sessão tem como objetivo impedir que alguém tente alterar a password de outro utilizador, sendo assim, é apenas permitido que altere a sua password e apenas caso se lembre da atual.

O token é verificado em todas as rotas (as rotas debaixo da pasta `public` que sejam para a pasta `scores`, `javascripts` e `pdfs` também estão protegidas), exceto no acesso ao login (`GET`) e no registo (`GET`'s e `POST`'s) sendo a função `isAuthenticated` responsável por perceber se o utilizador está autenticado (através do token). Já a função `havePermissions` tem como objetivo perceber se o utilizador tem permissões para aquela rota. Tanto as rotas visíveis ao utilizador como a API estão protegidas por estas duas funções, sendo que no caso em que é usado o `axios` para fazer pedidos internos, é-lhe passado o `cookie` enviado pelo utilizador, garantindo assim que é possível usar a API e que todas as rotas estão protegidas.

Quanto aos níveis dos utilizadores, o sistema apresenta três:

- Nível 1 (**Administrador**): Tem acesso a quase tudo (`GET`'s, `POST`'s, `PUT`'s e `DELETE`'s) sendo que a única coisa que não pode fazer é `POST` de obras e de utilizadores visto cada pessoa realizar o seu registo.

- Nível 2 (**Produtor**): Tem acesso a GETs de tudo exceto de utilizadores e pode fazer POSTs de obras, e POSTs, PUTs e DELETEs de instrumentos nas obras.
- Nível 3 (**Consumidor**): Tem a acesso a GETs de tudo exceto de utilizadores.

Contudo, os utilizadores que não possuam qualquer nível de permissão, ou seja não estejam registados, podem-se registar sendo que só terão acesso ao sistema assim que o administrador aprovar o registo. É também importante referir que caso um utilizador autenticado tente efetuar outro login ou registo, não lhe é permitido sendo redirecionado para a área principal do utilizador. Este redirecionamento é garantido pela função `authenticated` que verifica se o utilizador está autenticado.

## 5 Views

As *Views* variam consoante o tipo de utilizador tendo, assim, em conta de quem é proveniente o pedido (garantido através do `type` guardado na sessão). Para além disso, o sistema recorre ao uso de `jquery` de forma a tornar as páginas web mais funcionais, tais como o uso de filtros para filtrar o conteúdo a mostrar consoante o input do utilizador.

Por outro lado, por forma a mostrar os erros e/ou os sucessos por parte do utilizador, o sistema usa a package `connect-flash` como `middleware` que permite gravar nas sessões valores a apresentar. No fundo, isto permite que seja guardado o erro/sucesso a transmitir, ocorra o redirecionamento para uma página e, por fim, apresentar o erro/sucesso ao utilizador.

Na pasta **View** podemos encontrar sete sub-pastas correspondentes às *views* dos artigos, das *entries* (biblioteca de suporte), dos eventos, dos menus, das peças musicais, das estatísticas e, por fim, dos utilizadores. Para além das pastas, há ainda três ficheiros: a *view* dos erros, o *layout* básico, e um segundo *layout* que dá *extend* do primeiro.

```
doctype html
html
  head
    title iBanda
    link(rel='stylesheet', href='/stylesheets/w3.css')
    link(rel='shortcut icon' href='images/favicon.ico' type='image/icon')
    script(src="/javascripts/jquery-3.3.1.min.js")
    script.
      var url = "{url}"
    block scripts
  body
    block content
```

*Ficheiro layout.pug da pasta Views correspondente ao layout geral da aplicação.*

```
extends layout
block content
  .w3-container
    .w3-card-4
      header.w3-container.w3-grey
        block header
      .w3-container
        block body
      footer.w3-container.w3-grey
        address iBanda
```

*Ficheiro layout2.pug da pasta Views correspondente ao layout extendido da aplicação.*



A nível da interface, podemos ver nas figuras que se seguem alguns exemplos relativos às vistas disponíveis para um administrador.

A figura 3 ilustra um dos menus disponíveis, nomeadamente o menu das obras musicais no qual este pode consultar as várias obras disponíveis.

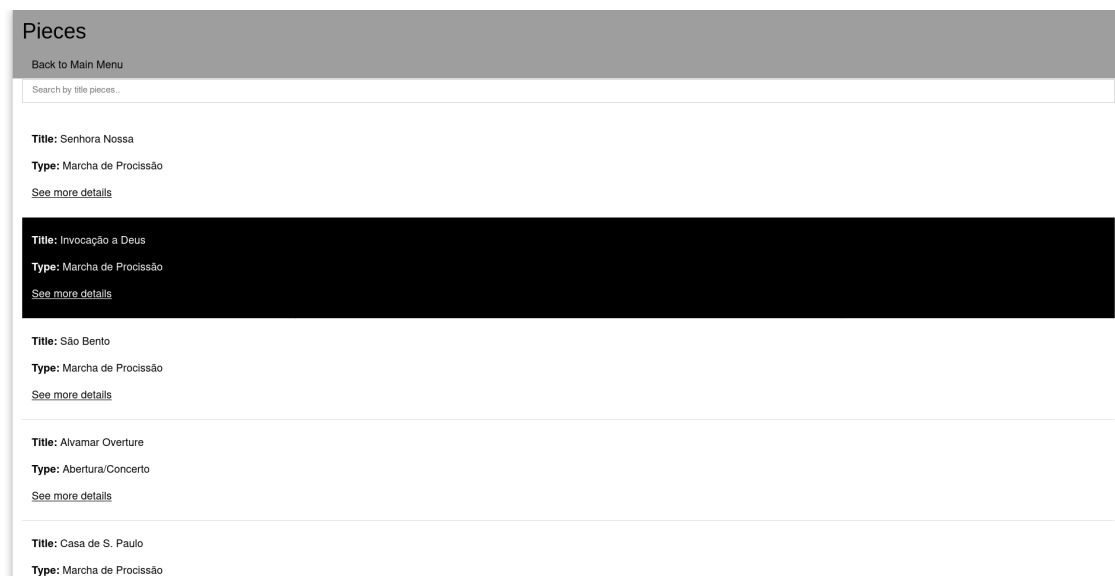


Figura 3: *Menu das obras musicais ("Pieces") disponível para um utilizador pertencente à categoria de "administrador".*

Assumindo que um administrador pretende consultar uma obra específica do menu acima mencionado, a vista resultante seria como a exemplificada na figura 4.

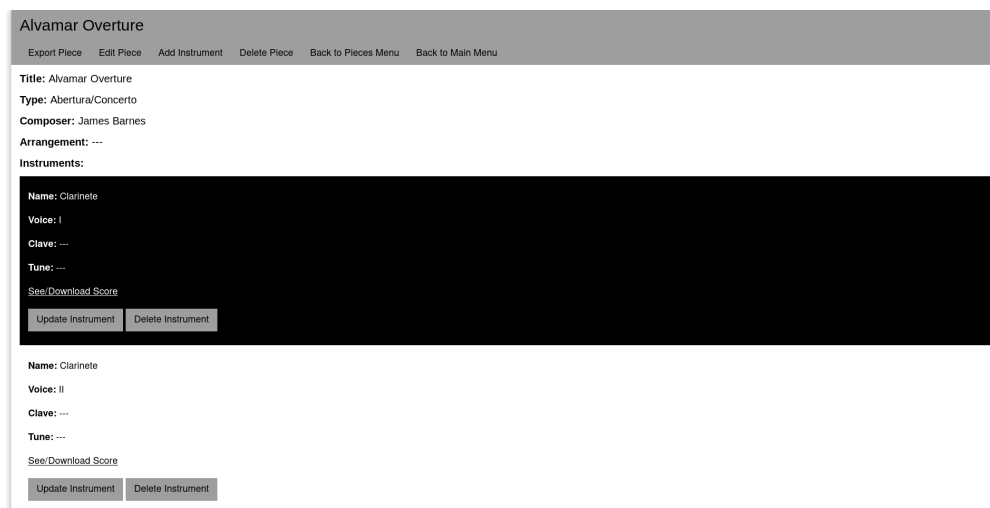


Figura 4: *Exemplo de uma obra disponível para um utilizador pertencente à categoria de "administrador".*

Por fim, caso um administrador pretenda consultar estatísticas relativas à aplicação, como por exemplo "User with most downloads" e "Piece with most views", ser-lhe-á apresentada uma vista como a da figura 5.

Statistics

Back to Main Menu

Total views	5
Total downloads	0
User with most views	Id: <a href="#">5c522b0e5116a27599a262bd</a> , with 5 views
User with most downloads	Id: <a href="#">5c522b0e5116a27599a262bd</a> , with 0 downloads
Piece with most views	Id: <a href="#">5c5381852b17132d71413ea</a> , with 2 views
Piece with most downloads	Id: <a href="#">5c5381852b17132d71413ea</a> , with 0 downloads

Search by name of users...

Name	Email	Type	Views	Downloads
root	root@root	Admin	5	0
lrm	email@piece	Producer	0	0

Blends

Figura 5: Menu das estatísticas disponível para um utilizador pertencente à categoria de "administrador".

## 6 Localização dos ficheiros armazenados

Os **ficheiros da biblioteca de suporte** (PDFs) estão todos na pasta `public/pdfs/` sendo que o nome de cada pdf é o `_id` gerado pelo mongoDB para representar o mesmo na base de dados mais a terminação do formato do ficheiro, ou seja `_id.pdf`.

Quanto aos **ficheiros correspondentes às obras**, são todos guardados na pasta `public/scores/`. Concretamente, cada obra tem uma pasta dedicada, sendo o nome dessa pasta o `_id` da obra (ou seja, a localização dos seus ficheiros) com id `_id` será `public/scores/_id/`.

As **sessões dos utilizadores** são guardadas em `sessions/`.

Os **logs do sistema** para além de serem mostrados na consola, são também guardados num ficheiro (`log.json`) usando o `morgan`, em que cada pedido é registado num formato `json`.

## 7 Gramáticas

Um das funcionalidades interessantes do sistema é a possibilidade de adicionar eventos e notícias de forma textual, isto é, introduzindo um texto estruturado com as informações necessárias à inserção da informação na base de dados. Esta opção está disponível na secção *News management* com a opção *Insert Articles by Grammar* e na secção *Agenda management* com a opção *Insert Events By Grammar*. Para esta poder ser completada com sucesso, isto é, a informação ser extraída corretamente e serem criados eventos/notícias, o texto introduzido é verificado quanto à sua correção sintática de acordo com as gramáticas elaboradas. Também é importante referir que estas funcionalidades apenas estão disponíveis para o administrador.

### 7.1 Agenda

Para a criação da funcionalidade de adicionar eventos *by grammar*, começamos por criar o ficheiro com a gramática de atributos utilizada na verificação. Nesse sentido, analisamos os *Schemas* disponíveis na pasta `Models`, nomeadamente os relativos ao artigos de notícias e aos eventos da agenda. Isto permitiu-nos saber que atributos incluir na gramática e também quais são ou não *required* para definir corretamente as produções.

A gramática de atributos criada tem o seu código escrito em *JavaScript* e possui 14 produções (`agenda`, `event`, `title`, `desc`, `local`, `time`, `start`, `end`, `startDate`, `startHour`, `endDate` e `endHour`). Fazemos algumas verificações, nomeadamente a correção temporal, isto é, se a data marcada para o fim do evento é efetivamente depois da data inicial (caso o evento seja num só dia, verifica se a hora de fim é posterior à hora de início).

Segue-se um excerto da gramática em questão.

```
grammar agenda;
```

```

agenda returns [var val, var errors]
@init {
    $val = []
    $errors = []
}
: 'AGENDA:' (
    event {
        if($event.error!="") $errors.push($event.error);
        else $val.push($event.val);
    } ';'
);

event returns[var val, var error]
@init {
    var existsDesc = false
    var existsLocal = false
}
: title (desc {existsDesc=true})? (local {existsLocal=true})? time
{
    if($time.ts.sdate>$time.te.edate){
        $val = ""
        $error = "Start date after end date on "+$title.val+" event!"
    }else{
        if ($time.ts.sdate==$time.te.edate && $time.ts.shour>$time.te.ehour
        ){
            $val = ""
            $error = "Start hour after end hour on "+$title.val+" event!"
        }else {
            $val = {
                title: $title.val,
                startDate: $time.ts.sdate,
                startHour: $time.ts.shour,
                endDate: $time.te.edate,
                endHour: $time.te.ehour
            }
            if (existsDesc) $val.desc = $desc.val
            else $val.desc = ""
            if (existsLocal) $val.local = $local.val
            else $val.local = ""
            $error = ""
        }
    }
}
;

```

*Excerto do ficheiro agenda.g4 da pasta grammars.*

No excerto acima é possível ver a verificação da data e a criação da mensagem de erro, quando esta se aplica. É ainda possível ver que a gramática permite a adição de vários eventos de uma só vez. Assim, o resultado de tentar adicionar os eventos do ficheiro abaixo seria inserção correta destes, visto que respeitam a estrutura definida na gramática sendo portanto um exemplo sintaticamente correto. Por outro lado, a nível semântico, também não acarreta problemas já que as datas/horas seguem as regras definidas.

AGENDA:

```
TITLE: 'Festa das Colheitas'
DESC: 'Celebração do mundo rural: agricultura, gastronomia, tradições.'
LOCAL: 'Vila Verde'
BEGINS: 2019-10-03 09:00
ENDS: 2019-10-07 23:00
;

TITLE: 'Santo Isidro'
BEGINS: 2019-09-30 10:00
ENDS: 2019-09-30 11:00
;
```

*Ficheiro agenda\_right.g4 da pasta grammars/test\_files.*

No entanto, o ficheiro abaixo não irá resultar na inserção do evento nele definido. Apesar de os campos obrigatórios se encontrarem todos definidos, a nível semântico é possível verificar que o *time* definido não faz sentido uma vez que a hora de fim do evento é posterior à hora de início (sendo que o evento é de um só dia). Assim, o evento não pode ser inserido e resulta na mensagem de erro *"Start hour after end hour on "Santo Isidro" event!"*

```
AGENDA:

TITLE: 'Santo Isidro'
BEGINS: 2019-09-30 10:00
ENDS: 2019-09-30 08:00
;
```

*Excerto do ficheiro agenda\_wrong.g4 da pasta grammars/test\_files.*

O exemplo abaixo também não pode ser inserido já que não contém dois dos campos obrigatórios (BEGINS: e ENDS:), nomeadamente data e hora de início e fim. Assim, falha devido a erros sintáticos, apesar do título, descrição e local estarem bem definidos.

```
AGENDA:

TITLE: 'Festa São Miguel'
DESC: 'Banda toca na procissão.'
LOCAL: 'Prado São Miguel'
;
```

*Excerto do ficheiro agenda\_wrong.g4 da pasta grammars/test\_files.*

Por fim, a tentativa de inserção do evento *'Festas das Colheitas'* irá falhar também devido erros sintáticos como o anterior. Efetivamente, todos os campos (TITLE, DESC, BEGINS, ENDS e LOCAL) estão corretamente definidos, e a datas/horas seguem as regras estipuladas. No entanto, não estão pela ordem esperada (definida na gramática), logo resultam em erros sintáticos.

AGENDA:

```
TITLE: 'Festa das Colheitas'
BEGINS: 2019-10-03 09:00
ENDS: 2019-10-07 23:00
DESC: 'Celebração do mundo rural: agricultura, gastronomia, tradições.'
LOCAL: 'Vila Verde'
;
```

*Excerto do ficheiro agenda\_wrong.g4 da pasta grammars/test\_files.*

## 7.2 News

A opção *Add Article By Grammar* foi concebida, tal como a sua semelhante dos eventos, tendo como base o seu ficheiro correspondente nos *Models*, concretamente, o *schema* definido no *article.js*. A gramática em questão possui 10 produções: *newspaper*, *news*, *titles*, *title*, *topics*, *topic*, *body*, *date*, *authors* e *author*.

A nível de verificações efetuadas, é conferida a data da notícia e, caso essa seja posterior ao dia atual, a inserção falha com o aviso *Date is in the future on X article*.

Segue-se um excerto da gramática.

```
grammar news;

newspaper returns [var val, var errors]
@init{
    $val = []
    $errors = []
}
: 'NEWS:' (news {
    if ($news.error != "") $errors.push($news.error);
    else $val.push($news.val);
} ';' )+
;

news returns [var val, var error]
@init{
    function today(){
        var today = new Date();
        var dd = today.getDate();
        var mm = today.getMonth() + 1; //January is 0!
        var yyyy = today.getFullYear();

        if (dd < 10) {
            dd = '0' + dd;
        }

        if (mm < 10) {
            mm = '0' + mm;
        }

        today = yyyy + '-' + mm + '-' + dd;
        return today;
    }
}
```

```

var existTopics = false
var existAuthors = false
}
: titles (topics {existTopics = true})? body date (authors {existAuthors = true})
?
{
  if(today()>=$date.val){
    $val = {
      title: $titles.titleOut,
      subtitle: $titles.subtitle,
      body: $body.val,
      date: $date.val
    }
    if(existTopics) $val.topics = $topics.val
    else $val.topics = []
    if(existAuthors) $val.authors = $authors.val
    else $val.authors = []
    $error = ""
  }else{
    $val = ""
    $error = "Date is in the future on " + $titles.titleOut + " article!"
  }
}
;

```

*Excerto do ficheiro news.g4 da pasta grammars.*

Segue-se um exemplo que resulta na adição à coleção dos artigos de notícias, visto que ambas as notícias definidas estão corretas a nível sintático e semântico, contendo a primeira todos os campos possíveis preenchidos e a segunda apenas os de preenchimento obrigatório (TITLE, BODY e DATE).

```

NEWS:

TITLE: 'Banda Filarmónica da UM'
SUBTITLE: 'Recorde de Concertos Ultrassado'
TOPICS: 'Universidade do Minho' , 'concertos'
BODY: 'A banda filarmónica da Universidade do Minho bateu este mês o recorde de
concertos por ano.'
DATE: 2018-12-27
AUTHORS: 'D. Barbosa' , 'JC Martins'
;

TITLE: 'A melhor Banda Filarmónica'
BODY: 'Campeã do Mundial de Bandas Filarmónicas.'
DATE: 2018-01-12
;

```

*Excerto do ficheiro news\_right.txt da pasta grammars/test\_files.*

No entanto, em alguns casos não é possível adicionar os artigos tal como pretendido. Um exemplo no qual isso acontece é quando ocorrem erros semânticos relativos à data. Exemplificando, não tem muito nexo escrever uma notícia com data posterior ao dia atual. Tal como mencionado previamente, este é um caso cuja verificação está definida na gramática. Seguindo esta lógica, a notícia abaixo (de título *"Banda Filarmónica da UM"*) não poderá ser adicionada antes do dia 27 de março de 2019.

```
NEWS:

TITLE: 'Banda Filarmónica da UM'
BODY: 'A banda filarmónica da Universidade do Minho bateu este mês o recorde de
concertos por ano.'
DATE: 2019-03-27
;
```

*Excerto do ficheiro news\_wrong.txt da pasta grammars/test\_files.*

Outro caso que não resulta numa correta inserção do artigo, está relacionado com erros sintáticos. No caso, a ausência do preenchimento de campos de natureza *required*, nomeadamente *DATE*.

```
NEWS:

TITLE: 'A melhor Banda Filarmónica'
BODY: 'Campeã do Mundial de Bandas Filarmónicas.'
AUTHORS: 'D. Barbosa' , 'JC Martins'
;
```

*Excerto do ficheiro news\_wrong.txt da pasta grammars/test\_files.*

Por fim, outro exemplo de erros sintáticos que impedem o funcionamento correto do processo *Add News By Grammar* é a troca das posições dos campos na notícia o que impede um reconhecimento adequado da produção. No caso abaixo ilustrado, o campo *SUBTITLE* que deveria estar imediatamente após o *TITLE* encontra-se a seguir ao campo *AUTHORS*, entre outras incoerências.

```
NEWS:

TITLE: 'Banda Filarmónica da UM'
DATE: 2018-12-27
AUTHORS: 'D. Barbosa' , 'JC Martins'
SUBTITLE: 'Recorde de Concertos Ultrassado'
TOPICS: 'Universidade do Minho' , 'concertos'
BODY: 'A banda filarmónica da Universidade do Minho bateu este mês o recorde de
concertos por ano.'
;
```

*Excerto do ficheiro news\_wrong.txt da pasta grammars/test\_files.*

### 7.3 Interligação entre Gramáticas e Web site

Tudo isto é possível devido ao *Gradle Build Tool*, um sistema de automação de compilação *open-source* com vários plugins, sendo que o plugin utilizado é o *java* de forma a gerar código javascript, em vez de código java, a ser utilizado pelo sistema (iBanda) desenvolvido.

```

apply plugin: 'java'

repositories {
    jcenter()
}

dependencies {
    runtime 'org.antlr:antlr4:4.5.2'
}

task generateAgendaParser(type:JavaExec) {
    main = 'org.antlr.v4.Tool'
    classpath = sourceSets.main.runtimeClasspath
    args = ['-Dlanguage=JavaScript', 'agenda.g4', '-o', 'agenda']
}

task generateNewsParser(type:JavaExec) {
    main = 'org.antlr.v4.Tool'
    classpath = sourceSets.main.runtimeClasspath
    args = ['-Dlanguage=JavaScript', 'news.g4', '-o', 'news']
}

```

*Ficheiro build.gradle da pasta Grammars [1].*

Para além disso, já no sistema desenvolvido é usado a package `antlr4/index` disponível em *JavaScript*, por forma a utilizar o código gerado pelo Gradle.

O excerto seguinte de código realiza a verificação léxica e sintática do input para o caso da agenda:

```

var NewsLexer = require('../grammars/news/newsLexer').newsLexer
var NewsParser = require('../grammars/news/newsParser').newsParser
var NewsListener = require('../grammars/news/newsListener').newsListener

.
.
.

//recebe o input e transforma-o num tipo reconhecido pelo antlr4
var chars = new antlr4.InputStream(req.body.grammar);
//cria o lexer a usar, a partir do código gerado pelo gradle
var lexer = new AgendaLexer(chars);
var tokens = new antlr4.CommonTokenStream(lexer);
//cria o parser a usar, a partir do código gerado pelo gradle
var parser = new AgendaParser(tokens);

//variável que irá guardar os erros léxicos gerados pela gramática
var log = []

//trocado o console.error por forma a capturar os erros gerados pelo analisador léxico
var exLogError = console.error
console.error = function(msg) {
    log.push(msg)
}

parser.buildParseTrees = true;
//cria-se a árvore ao chamar a regra onde se começa (neste caso começa-se pela regra agenda)
var tree = parser.agenda();
//criação do listener
var agendaListener = new AgendaListener();
//é percorrida a árvore criada com o listener associado (no listener podia ser associado
código à entrada e à saída das regras, o listener usado é o default que não possui nenhum
código nessas fases)

```



```
antlr4.tree.ParseTreeWalker.DEFAULT.walk(agendaListener, tree);

//reset do console.error
console.error = exLogError
```

*Exemplo de Interligação para a Agenda entre o código gerado pelo ANTLR a partir da gramática produzida e o sistema desenvolvido. [4, 2, 3]*

De forma a obter o que foi construído pelo parser, existem dois atributos sintetizados na regra inicial a partir dos quais é possível obter os artigos (**var val**) e os erros (**var errors**). Estes atributos são acessíveis após percorrer a árvore fazendo **tree.val** para obter **val** e **tree.errors** para obter o **errors**.

Tendo isto em conta, caso hajam erros quer léxicos (**var log**) quer sintáticos (**var errors**), serão apresentados ao user. Caso não hajam erros, são inseridos na base de dados os eventos.

Para as *news* é feito de forma semelhante, mudando apenas o facto de que em vez de se usarem os *lexer's*, *parser's* e *listener's* do **agenda** usa-se do *news*, bem como a regra inicial que é **newspaper** em vez de **agenda**.

## 8 Instalação

Para proceder à instalação do sistema de forma correta, é necessário que a máquina em questão tenha instaladas as seguintes ferramentas: **Java 8**, **Gradle**, **npm** e **mongoDB**. Após a instalação das dependências referidas, é possível iniciar a instalação, correndo o seguinte conjunto de passos (assumindo que possui todos os ficheiros e que a directoria atual seja a pasta *iBanda*):

```
$ cd grammars
$ gradle generateAgendaParser
$ gradle generateNewsParser
$ cd ..
$ npm install
```

Por fim, para correr o sistema, são ainda necessários os comandos:

```
$ mongod &
$ npm start
```

Por forma a automatizar este processo foi criada uma script com 4 modos. No primeiro, (**./install.sh p**), é feito o clone do repositório, esperado que o user mude o *url* em **app.js** (bem como a porta em **bin/www**), a chave privada, a chave pública e, opcionalmente, a password do *admin* (**root@root**), correndo no fim os passos anteriormente enunciados. No segundo, **./install.sh i**, são realizados os passos anteriormente enunciados, sem realizar clone, e sem esperar pelo utilizador. Os últimos dois modos criam a pasta **~/Downloads/mongoDB** de modo a correr o mongoDB da forma **mongod --dbpath ~/Downloads/mongoDB**, no qual as BDs são guardadas nessa pasta e estes dois modos servem mais para testes. Com **./install t**, compila-se as gramáticas (gradle) e começa o mongoDB e o sistema sem fazer a instalação de packages npm dos quais o sistema depende. Por fim, o **./install r** apenas corre o mongoDB e o sistema.

```
#!/bin/bash

if [ $# -eq 1 ]; then
    if [ $1 = "i" ]; then
        #start mongoDB
        mongod &

        #generate grammars
        cd grammars
        gradle generateAgendaParser
        gradle generateNewsParser
```

```
#Install packages dependencies
cd ..
npm install

#Start
npm start
```

*Excerto do ficheiro install.sh da pasta iBanda que possui os vários modos de instalação.*

## 9 Conclusões

As aplicações *web* que podem ser criadas hoje em dia têm um enorme potencial para ajudar negócios, pessoas e organizações. Neste caso, o público alvo são bandas filarmónicas, no entanto, o sistema é facilmente adaptável a uma grande variedade de contextos. A nível de tecnologias, ferramentas e conceitos de que tiramos partido, destacam-se os modelos *MVC* da aplicação, *OAIS* do repositório, e o *npm*.

Com este trabalho, o grupo conseguiu aprimorar as suas competências no uso de *npm*, ferramentas novas como *Gradle Build Tool*, mas também a nível de programação em *JavaScript*. Conseguiu ainda desenvolver pensamento crítico devido aos desafios que foram surgindo (como por exemplo, a ligação das gramáticas ao sistema) e, em suma, pôr em prática os conhecimentos adquiridos ao longo do semestre.

No entanto, o trabalho apresenta alguns pontos a melhorar, entre os quais, avisar por email quando a conta é aprovada, quando forem alteradas as suas permissões ou quando a conta é apagada. Por outro lado, permitir a inserção de texto na parte gramatical a partir de um ficheiro seria também uma funcionalidade extra interessante. Por fim, o outro ponto a melhorar seria o acréscimo da funcionalidade da enciclopédia do material armazenado que consiste na busca de informação à *internet* sobre os autores e as obras armazenadas.

Assim, em conclusão, fazemos uma avaliação positiva a nível das funcionalidades implementadas e objetivos cumpridos.

## Referências

- [1] *ANTLR and the web: a simple example*. <https://tomassetti.me/antlr-and-the-web/>. Acedido em 23/01/2019.
- [2] *Compiler in JavaScript using ANTLR*. <https://medium.com/dailyjs/compiler-in-javascript-using-antlr-9ec53fd2780f>. Acedido em 23/01/2019.
- [3] *JavaScript - How do I run the generated lexer and/or parser?* <https://github.com/antlr/antlr4/blob/master/doc/javascript-target.md>. Acedido em 23/01/2019.
- [4] *The ANTLR Mega Tutorial*. <https://tomassetti.me/antlr-mega-tutorial/#setup-antlr-with-javascript>. Acedido em 23/01/2019.

## A Gramática da Agenda

```
grammar agenda;

agenda returns [var val, var errors]
@init {
    $val = []
    $errors = []
}
: 'AGENDA:' (
    event {
        if($event.error!="") $errors.push($event.error);
        else $val.push($event.val);
    } ';'
)+;

event returns[var val, var error]
@init {
    var existsDesc = false
    var existsLocal = false
}
: title (desc {existsDesc=true})? (local {existsLocal=true})? time
{
    if($time.ts.sdate>$time.te.edate){
        $val = ""
        $error = "Start date after end date on \"" + $title.val + "\" event!"
    }else{
        if ($time.ts.sdate==$time.te.edate && $time.ts.shour>$time.te.ehour ){
            $val = ""
            $error = "Start hour after end hour on \"" + $title.val + "\"
                event!"
        }else {
            $val = {
                title: $title.val,
                startDate: $time.ts.sdate,
                startHour: $time.ts.shour,
                endDate: $time.te.edate,
                endHour: $time.te.ehour
            }
            if (existsDesc) $val.desc = $desc.val
            else $val.desc = ""
            if (existsLocal) $val.local = $local.val
            else $val.local = ""
            $error = ""
        }
    }
}
;

title returns[var val]
: 'TITLE:' TEXT {$val = $TEXT.text.substring(1, $TEXT.text.length-1)};

desc returns[var val]
: 'DESC:' TEXT {$val = $TEXT.text.substring(1, $TEXT.text.length-1)};

local returns[var val]
: 'LOCAL:' TEXT {$val = $TEXT.text.substring(1, $TEXT.text.length-1)};
```

```

time returns[var ts, var te]
: s = start e = end
{
    $ts = {
        sdate: $s.sd,
        shour: $s.sh
    }
    $te = {
        edate: $e.ed,
        ehour: $e.eh
    }
};

start returns[var sd, var sh]
: 'BEGINS:' d = startDate h = startHour
{
    $sd = $d.val
    $sh = $h.val
};

end returns[var ed, var eh]
: 'ENDS:' d = endDate h = endHour
{
    $ed = $d.val
    $eh = $h.val
};

startDate returns[var val]: DATE {$val = $DATE.text};

startHour returns[var val]: HOUR {$val = $HOUR.text};

endDate returns[var val]: DATE {$val = $DATE.text};

endHour returns[var val]: HOUR {$val = $HOUR.text};

/* Definição do Analisador Léxico */
TEXT: (['"'] ~(['"])* ['"]);

fragment DIGIT: [0-9];

DATE: DIGIT DIGIT DIGIT DIGIT '-' DIGIT DIGIT '-' DIGIT DIGIT;

HOUR: DIGIT DIGIT ':' DIGIT DIGIT;

Separator: ( '\r'? '\n' | ' ' | '\t' )+ -> skip;

```

## B Gramática das News

```
grammar news;

newspaper returns [var val, var errors]
@init{
    $val = []
    $errors = []
}
    : 'NEWS:' (news {
        if($news.error!="") $errors.push($news.error);
        else $val.push($news.val);
    } ';')+
    ;

news returns [var val, var error]
@init{
    function today(){
        var today = new Date();
        var dd = today.getDate();
        var mm = today.getMonth() + 1; //January is 0!
        var yyyy = today.getFullYear();

        if (dd < 10) {
            dd = '0' + dd;
        }

        if (mm < 10) {
            mm = '0' + mm;
        }

        today = yyyy + '-' + mm + '-' + dd;
        return today;
    }
    var existTopics = false
    var existAuthors = false
}
: titles (topics {existTopics = true})? body date (authors {existAuthors = true})?
{
    if(today()>=$date.val){
        $val = {
            title: $titles.titleOut,
            subtitle: $titles.subtitle,
            body: $body.val,
            date: $date.val
        }
        if(existTopics) $val.topics = $topics.val
        else $val.topics = []
        if(existAuthors) $val.authors = $authors.val
        else $val.authors = []
        $error = ""
    }else{
        $val = ""
        $error = "Date is in the future on \" + $titles.titleOut + "\" article!"
    }
}
;
```

```

titles returns [var titleOut, var subtitle]
@init{
    var subExists = false
}
: 'TITLE:' t=title ('SUBTITLE:' s=title {subExists = true})?
{
    $titleOut = $t.val
    if(subExists) $subtitle = $s.val
    else $subtitle = ""
}
;

title returns [var val]
: TEXT {$val = $TEXT.text.substring(1, $TEXT.text.length-1)}
;

topics returns [var val]
@init{
    $val = []
}
: 'TOPICS:' t1=topic {$val.push($t1.val)} (',' t2=topic {$val.push($t2.val)})*
;

topic returns [var val]
: TEXT {$val = $TEXT.text.substring(1, $TEXT.text.length-1)}
;

body returns [var val]
: 'BODY:' TEXT {$val = $TEXT.text.substring(1, $TEXT.text.length-1)}
;

date returns [var val]
: 'DATE:' DATE {$val = $DATE.text}
;

authors returns [var val]
@init{
    $val = []
}
: 'AUTHORS:' a1=author {$val.push($a1.val)} (',' a2=author {$val.push($a2.val)})*
;

author returns [var val]
: TEXT {$val = $TEXT.text.substring(1, $TEXT.text.length-1)}
;

/* Definição do Analisador Léxico */
TEXT: (['"] ~(['"])* ["]);

fragment DIGIT: [0-9];

DATE: DIGIT DIGIT DIGIT DIGIT '-' DIGIT DIGIT '-' DIGIT DIGIT;

Separator: ( '\r'? '\n' | ' ' | '\t' )+ -> skip;

```