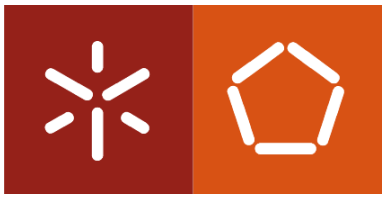

Escola de Engenharia da Universidade do Minho
Departamento de Informática
Mestrado Integrado em Engenharia Informática
Processamento e Representação de Informação

iBanda - Arquivo Digital Musical

João Gomes, A74033
Tiago Fraga, A74092

14 de Janeiro de 2019

Resumo

Trabalho realizado no âmbito da unidade curricular Processamento e Representação de Informação do perfil de Processamento de Linguagens e Conhecimento do 4º ano do Mestrado Integrado em Engenharia Informática.

O projeto consiste no desenvolvimento de uma Aplicação *Web* que implemente um repositório digital de obras musicais e respetivas partituras.

Este repositório teve de respeitar o modelo de referência internacional `OAIS` (*Open Archive Information System*)

Conteúdo

Capítulo 1

Introdução

Neste projecto, foi desenvolvida uma aplicação web que implementasse um repositório de arquivo de musica digital seguindo o modelo `OAIS` (Online Archive Information System).

Este modelo foi desenvolvido com o intuito de facilitar o consenso relativo aos requisitos para um repositório conseguir preservar a informação digital.

No nosso caso além deste modelo, temos várias funcionalidades que o sistema tem de executar, tais como :

- Gestão de uma Agenda de Eventos;
- Gestão de Notícias;
- Gestão de Utilizadores/Músicos da Banda;
- Gestão de Reportórios;
- Gestão de uma biblioteca de suporte.

Para desenvolver tal aplicação usamos o `Node JS` como ferramenta de desenvolvimento, `MongoDB` como sistema de Base de Dados onde armazenamos a maior parte dos dados do sistema.

O relatório em questão é constituído pelos seguintes temas que em seguida iremos apresentar:

- Criação e desenvolvimento de uma gramática de suporte à agenda de eventos..
- Funcionamento da aplicação com especial ênfase às funcionalidades do *back-end* e do *front-end*.
- Conclusões obtidas, bem como, pequenas anotações para a continuação deste trabalho.

Capítulo 2

Gramática

Neste capítulo explicamos como criamos a gramática, e qual a sua função.

2.1 Função

A gramática criada têm como função receber um ficheiro de texto com uma lista de eventos e criar outro ficheiro desta vez em *JSON* de modo a ser mais tarde inserido via Aplicação *Web* na base de dados criada.

A forma como desenvolvemos a gramática permite inserir mais que um evento, e permite ainda filtrar os eventos que são válidos dos inválidos, não descartando o ficheiro por completo. Sendo aprofundada esta parte mais a frente.

2.2 Estrutura

De forma a ser mais percetível apresentamos as regras principais que constituem a nossa gramática :

```
agenda : 'Agenda' '---' eventos
       ;
eventos: (evento '.')+
       ;
evento : 'Evento' '---' data ',' horario? ',' tipo ',' designacao? ','
          local? ',' informacao? ','
       ;
```

Com estas regras e as restantes fomos capazes de definir o evento, tal como representado no modelo criado na nossa aplicação.

Depois de definida todas as regras, definimos classes `Java` para armazenar os eventos válidos, bem como estruturas de dados.

Tendo toda a estrutura da gramática montada, acrescentamos ações semânticas e condições de contexto as produções bem como atributos herdados e sintetizados. De modo a validar o ficheiro de texto inserido, adicionar a informação válida nas estruturas de dados e por fim escrever todos os eventos válidos num ficheiro `JSON`, que depois de carregado na Aplicação é lido e adicionado a base de dados.

2.3 Ficheiro Input

Nesta secção vamos demonstrar tipos de eventos aceites pelo nosso sistema e tipos de eventos considerados invalidos.

Começamos por demonstrar um exemplo válido:

```
1 Agenda——Evento——Data:(01,02,2019),HoradeInicio:(20:45)–HoradeFim:(21:45),
    Tipo:Concerto de Bandas Filamornicas,Designacao:Concerto Sociedade de
    Bandas,Local:Braga,Informacao:Bilhete 5 euros.
```

Em seguida temos um exemplo de um evento inválido, visto o ano na data não ser maior que 2019:

```
1 Agenda——Evento——Data:(01,02,2017),HoradeInicio:(20:45)–HoradeFim:(21:45),
    Tipo:Concerto de Bandas Filamornicas,Designacao:Concerto Sociedade de
    Bandas,Local:Braga,Informacao:Bilhete 5 euros.
```

Agora temos em que a data inserida tem corresponde ao dia 30 do mês de Fevereiro, como Fevereiro só tem 28 ou 29 dias em caso de ano bissexto, o evento é invalido.

```
1 Agenda——Evento——Data:(30,02,2019),HoradeInicio:(20:45)–HoradeFim:(21:45),
    Tipo:Concerto de Bandas Filamornicas,Designacao:Concerto Sociedade de
    Bandas,Local:Braga,Informacao:Bilhete 5 euros.
```

Outro caso evidenciado pela gramática criada é um evento em que a hora de inicio é superior a hora do fim.

```
1 Agenda——Evento——Data:(01,02,2019),HoradeInicio:(21:45)–HoradeFim:(19:45),
    Tipo:Concerto de Bandas Filamornicas,Designacao:Concerto Sociedade de
    Bandas,Local:Braga,Informacao:Bilhete 5 euros.
```

2.4 Resultado

Nesta secção mostramos um input de texto com vários eventos e a transformação do mesmo pela gramática:

- **Input:** Ficheiro de Input criado.

```
Agenda——Evento——Data:(01,02,2019),HoradeInicio:(20:45)–HoradeFim
    :(21:45),Tipo:Concerto de Bandas Filamornicas,Designacao:Concerto
    Sociedade de Bandas,Local:Braga,Informacao:Bilhete 5 euros.Evento——
    Data:(04,04,2019),HoradeInicio:(15:53)–HoradeFim:(17:53),Tipo:Desfile
    de Bandas,Designacao:Desfile pela cidade,Local:Braga,Informacao:
    Gratuito.Evento——Data:(03,05,2020),HoradeInicio:(15:53)–HoradeFim
    :(17:53),Tipo:Concerto Sociedade Boa Uni o Alhadense,Designacao:
    Concerto Cultural,Local:Figueira da Foz,Informacao:M sica ao Ar livre
    . Evento——Data:(08,12,2019),HoradeInicio:(20:53)–HoradeFim:(21:53),
    Tipo:Concerto de Caridade,Designacao:Concerto no teatro,Local:Vila
    Real,Informacao:Pre o do bilhete reverte para ajudar associa  o.
```

- **Output:** Resultado final ficheiro `JSON` com o seguinte formato:

```
[{"data": "31/12/2019","horario":{"hinicio":"12:53","hfim":"15:53"},"tipo
    ":"Concerto","designacao":"Concerto","local":"Braga","informacao":"
    nada"},
{"data": "2/2/2020","horario":{"hinicio":"15:53","hfim":"17:53"},"tipo":"
    ConcertoLista","designacao":"ConcertoL","local":"Braga","informacao":"
    nada"},
{"data": "1/1/2020","horario":{"hinicio":"15:53","hfim":"17:53"},"tipo":"
    Ok","designacao":"f","local":"Braga","informacao":"nada"},
{"data": "3/3/2020","horario":{"hinicio":"15:53","hfim":"17:53"},"tipo":"
    S","designacao":"e","local":"Braga","informacao":"nada"},
{"data":"4/4/2020","horario":{"hinicio":"15:53","hfim":"17:53"},"tipo":"d
    ","designacao":"h","local":"Braga","informacao":"nada"}]
```

Capítulo 3

Aplicação Web

3.1 Estrutura

A aplicação foi desenvolvida segundo uma *API Rest*, sendo utilizada para o efeito a ferramenta *Express*.

Ao logo do desenvolvimento da mesma, e de forma a estruturamos a divisão do trabalho pelos criadores, dividimos a aplicação em duas etapas.

A primeira etapa é constituída pelo *Back-End*. Aqui é onde estão definidas as seguintes ferramentas:

- **Models** - Onde está definido os *Schemas* das coleções que vão ser guardadas na base de dados.
- **Controllers** - Onde estão definidas as *queries* sobre a base de dados de forma a adquirir a informação pretendida.
- **Routes/Api** - Onde estão definidas as rotas que invocam os *controllers*.

A segunda etapa é constituída pelo *Front-End*. Esta etapa é formada pelas seguintes ferramentas:

- **Routes** - Aqui estão definidas as rotas que são invocadas pelas páginas, sendo também a ponte para as rotas que invocam a *API* de dados.
- **Views** Aqui estão definidas as páginas apresentadas aos utilizadores no *browser*.

3.2 Funcionalidades

3.2.1 Administrador

O administrador é o utilizador que mais funcionalidades tem disponíveis dentro da aplicação. Entre elas, destacamos:

- Registo, remoção e atualização de utilizadores;
- Listagem de utilizadores;
- Listagem, remoção e exportação de obras;
- Registo, remoção e atualização de noticias;
- Listagem de noticias;
- Registo, remoção, atualização e exportação de eventos;
- Listagem de eventos;
- Registo de eventos a partir do ficheiro gerado pela gramática;
- Visualização de estatisticas de uso da aplicação.

3.2.2 Produtor

O produtor é o utilizador, que a seguir ao administrador, tem mais funcionalidades disponveis. Entre elas, destacamos:

- Registo, listagem, remoção e exportação de obras;
- Listagem de eventos;

3.2.3 Consumidor

Por outro lado, o consumidor é o utilizador que tem menos funcionalidades na aplicação, apenas podendo visualizar e exportar conteúdo. Em suma, as funcionalidades disponíveis sao:

- Listagem, remoção e exportação de obras;
- Listagem de eventos;

3.3 Dependências

O ficheiro *package.json* contem toda a meta informação à cerca da aplicação a que está associado, sendo que é composto por diferentes diretivas que informam o *NPM* de como tratar certos módulos ou pacotes.

As diretivas referidas podem ser obrigatórias, como o "name"e a "version", ou opcionais, como a "description"e a "dependencies". Dentro destas dependências gostaríamos de referir trˆes em particular:

- **mongoose** - Para a connecção a base de dados MongoDB
- **bcrypt** - Para encriptar as passwords na base de dados.
- **passport** - Para tratar dos processos de autenticação no sistema.
- **axios** - Para fazer a ponte entre a *API* de dados e as páginas.
- **express-easy-zip** - Para compactar uma pasta num ficheiro *ZIP*.
- **extract-zip** - Para descompactar um ficheiro *ZIP*.

3.4 Modelo de Dados

De modo a desenvolvermos a nossa aplicação *web* definimos os `Models`, isto é os modelos da nossa Base de Dados, possuindo assim um esquema de como a informação iria ser armazenada.

3.4.1 Modelo de Dados

Desta forma definimos os seguintes `Models`:

- **Model de Utilizador:**
  A informação relativa aos utilizadores é guardada com os seguintes atributos, sendo `_id` relativo ao *username* do Utilizador.

```
1            _id: {type: String,required: true},
2            password: {type: String, required: true},
3            name: {type: String, required: true},
4            email: {type: String, required: true,unique: true},
5            userType: {type: String, required: true}
```

- **Model Obra:**

  O `Model` da obra é o mais complexo de todo o nosso sistema, visto possuir dois `Sub-Schemas`, verificando-se em seguida.

```
_id: {type: String, required: true},
titulo: {type: String, required: true},
tipo: {type: String, required: true},
compositor: {type: String, required: true},
arranjo: {type: String},
instrumentos: [InstrumentoSchema]
```

  Como se pode verificar o atributo `instrumentos` é um `JSON array` composto pelo `Sub-Schema` `InstrumentoSchema` que é constituído da seguinte forma:

  - **Sub-Schema InstrumentoSchema:**

```
nome: {type: String},
partitura: {type: PartituraSchema}
```

    Como se pode observar este `Sub-Schema` apresenta um `Sub-Schema` denominado `PartituraSchema` de modo a representar o atributo partitura, em seguida mostramos este `Sub-Schema`

  - **Sub-Schema PartituraSchema:**

```
path: {type: String},
voz: {type: String},
clave: {type: String},
afinacao: {type: String}
```

    Por ultimo temos este `Sub-Schema`, composto pelos atributos mostrados a cima, com estes dois `Sub-Schemas` conseguimos ter toda a informação das obras bem representada.

- **Model Noticia:**

  Depois de temos representados os Utilizadores e as Obras, representamos as Noticias sendo o seu `Model` definido da seguinte forma:

```
_id: {type: Number, required: true},
titulo: {type: String, required: true},
corpo: {type: String, required: true},
```

```
4                 data: {type: String, required: true },
5               visibilidade: {type: Boolean, required: true},
```

Sendo o atributo *_id* criado no processo de inserção da notícia, através da incrementação do maior valor na base de dados.

- **Model Evento:**

Por último definimos o tipo de representação dos Eventos, em que é constituído por um `Sub-Schema`, semelhante ao `Sub-Schema` partitura da obra.

```
1               _id:{type: Number, required: true},
2               data: {type: String, required: true},
3               horario: {type: HorarioSchema},
4               tipo: {type: String, required: true},
5               designacao: {type: String},
6               local: {type: String},
7               informacao: {type: String}
```

Como podemos observar o atributo *horario* é composto por um `Sub-Schema` representado em seguida:

– **Sub-Schema HorarioSchema:**

```
1                   hinicio: {type: String},
2                   hfim: {type: String}
```

Sendo composto pela hora de inicio e fim do evento, ficando desta forma o evento totalmente representado.

3.4.2 Controllers

Depois de definidos todos os modelos de dados, tivemos de definir os `Controllers` de modo, a conseguir ler,inserir, eliminar e atualizar a informação guardada na base de dados.

Segue em seguida um exemplo de cada tipo de controlador para um determinado modelo.

- **Ler:**

  Temos como exemplo a obtenção de todos os utilizadores da nossa base de dados:

```
1         Users.listar = () => {
2         return User
3             .find()
4             .exec()
5         }
```

- **Inserir:**

  Em seguida temos a inserção de um utilizador:

```
1         Users.inserir = async u => {
2         var hash = await bcrypt.hash(u.password, 10)
3         u.password = hash
4         var user = new User({
5             _id: u.username,
6             password: u.password,
7             name: u.nome,
8             email: u.email,
9             userType: u.userType
10        })
11        return User.create(user)
12    }
```

- **Eliminar:**
  A operação de eliminação de um utilizador é feita da seguinte forma:

```
1          Users.remove = username =>{
2              return User.findOneAndRemove({_id: username},(erro,doc) =>{
3                  if(!erro){
4                      console.log('Utilizador removido com sucesso')
5                  }
6                  else{
7                      console.log('Nao consegui remover utilizador')
8                  }
9              })
10         }
```

- **Atualizar:**
  Por último temos o exemplo da atualização de um utilizador.

```
Noticias.atualiza = n =>{
    return Noticia.findOneAndUpdate({_id:n.id},{$set:{titulo:n.titulo,
        corpo: n.corpo, data: n.data,visibilidade:true}},{new: true},(erro,doc)=>{
        if(!erro){
        }
        else{
            console.log('Nao consegui atualizar utilizador')
        }

    })
}
```

3.5 Armazenamento de dados no FileSystem

Para garantir que todos os ficheiros que são colocados na aplicação, e de modo a conseguir mais tarde exportar, guardamos esses ficheiros no *FileSystem*.

3.5.1 Inserção de Obras

A inserção de obras é realizada através de um *upload* de um ficheiro *ZIP*, este ficheiro tem de obedecer a uma estrutura predefinida, isto é, tem de conter um ficheiro com o nome iBanda-SIP

em *JSON* e com a descrição da obra segundo o nosso modelo de dados. E todos os ficheiros referenciados por ele têm de estar numa pasta. Caso não obedeça a esta estrutura o ficheiro não é aceite.

Depois de validado a obra contida no iBanda-SIP é armazenada na base de dados e os ficheiros no *path*: public/catalogo/id_obra sendo este _id referenciado no ficheiro.

3.5.2 Exportação de Obras

Com o auxilio do *path* criado no momento da inserção, nós conseguimos facilmente saber quais os ficheiros de determinada obra. Sabendo o caminho é criado um *ZIP* com a mesma estrutura de quando é inserido e enviado ao cliente.

3.5.3 Exportação de Eventos

A exportação de eventos também é realizada através de um ficheiro *ZIP* enviado ao cliente, mas primeiro é criada um ficheiro em *JSON* com todos os eventos presentes na base de dados, esse ficheiro *JSON* e *ZIP* possuem como nome a data da sua exportação, permitindo ao administrador verificar todos os eventos que introduziu ou eliminou até ao momento. Este ficheiro a ser exportado é guardado também no *FileSystem*, mantendo a sua persistência.

3.5.4 Estatística de utilização

De modo a mantermos a estatística em relação ao número visualizações e *downloads* criamos um ficheiro *JSON*, armazenado no FileSystem no path: public/data/logs/logs.json com as seguintes características:

```
{
        "total": {
                "visualizacoes": 0,
                "downloads": 0
        },
        "produtor": {
                "visualizacoes": 0,
                "downloads": 0
        },
        "consumidor": {
                "visualizacoes": 0,
                "downloads": 0
        }
```

```
14 }
```

Este ficheiro sempre que acontece uma visualização da lista de obras ou *donwload* é incrementado o valor de total no campo correspondente, se for uma visualização ou *download* nas páginas referentes as listas de obras por um consumidor ou produtor é incrementado também o campo correspondente.

Assim permitimos que o administrador mantenha uma informação acerca destes parâmetros de estatística.

3.6 Vistas

Neste capitulo apresentamos a interface da aplicação. Iremos apresentar imagens de várias páginas para os vários tipos de utilizadores que podem interagir com a mesma.

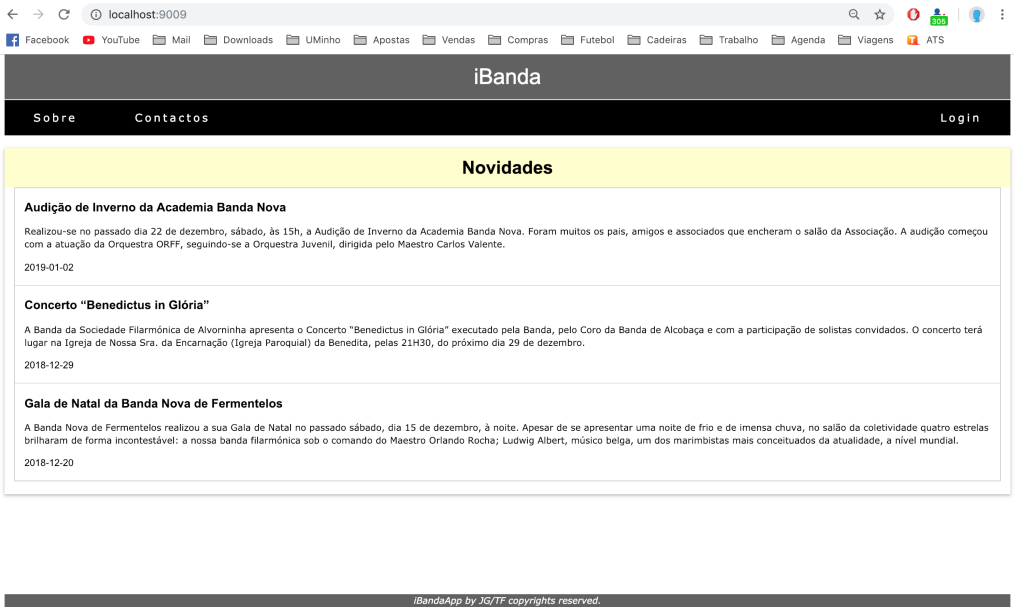

Figura 3.1: Página Inicial da Aplicação.

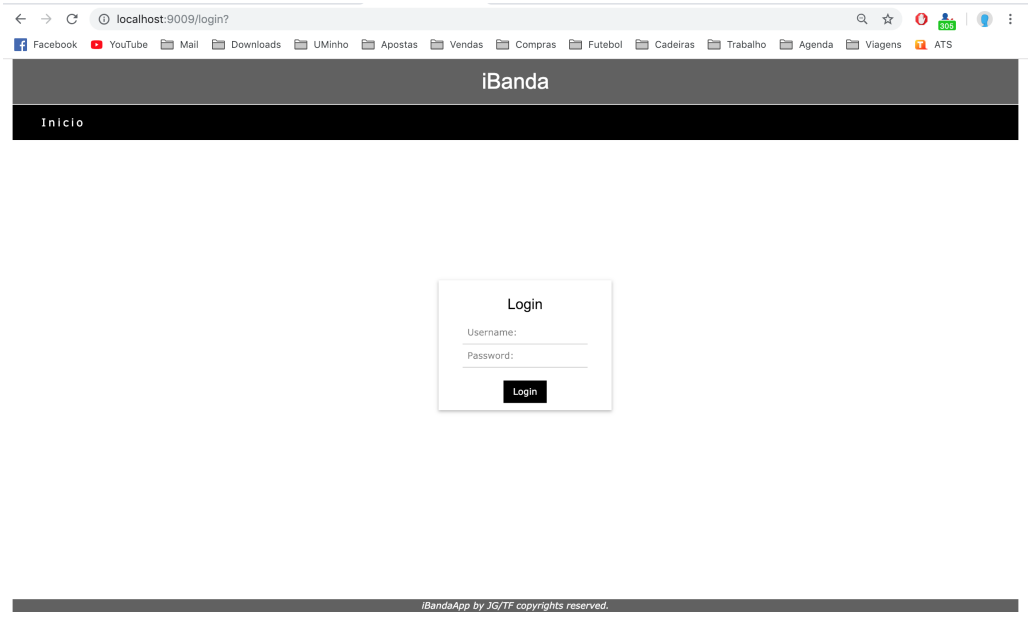

Figura 3.2: Página de login.

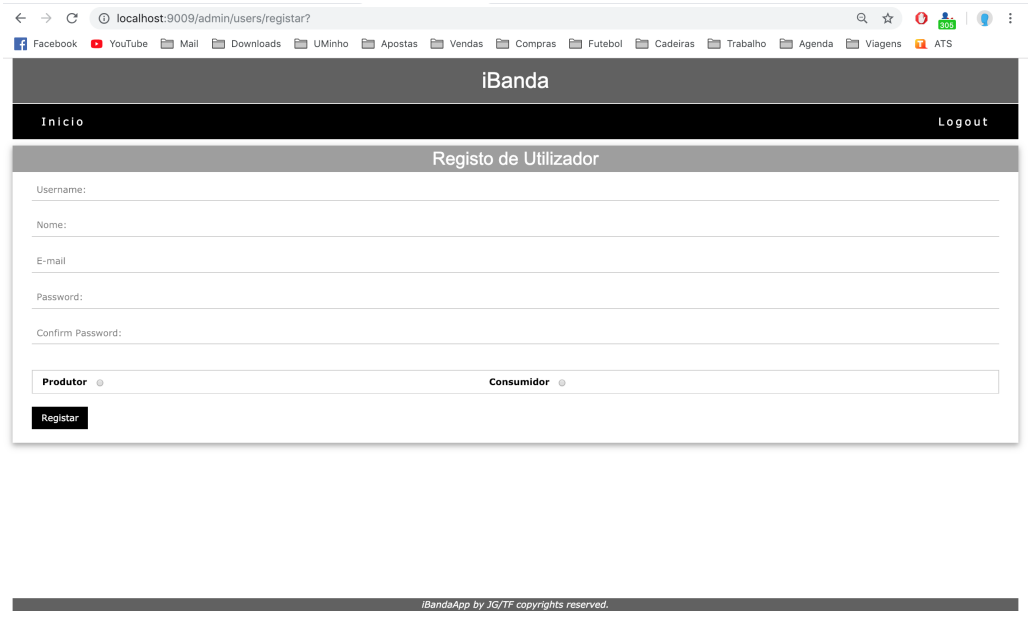

Figura 3.3: **Administrador** - Registo de Utilizador.

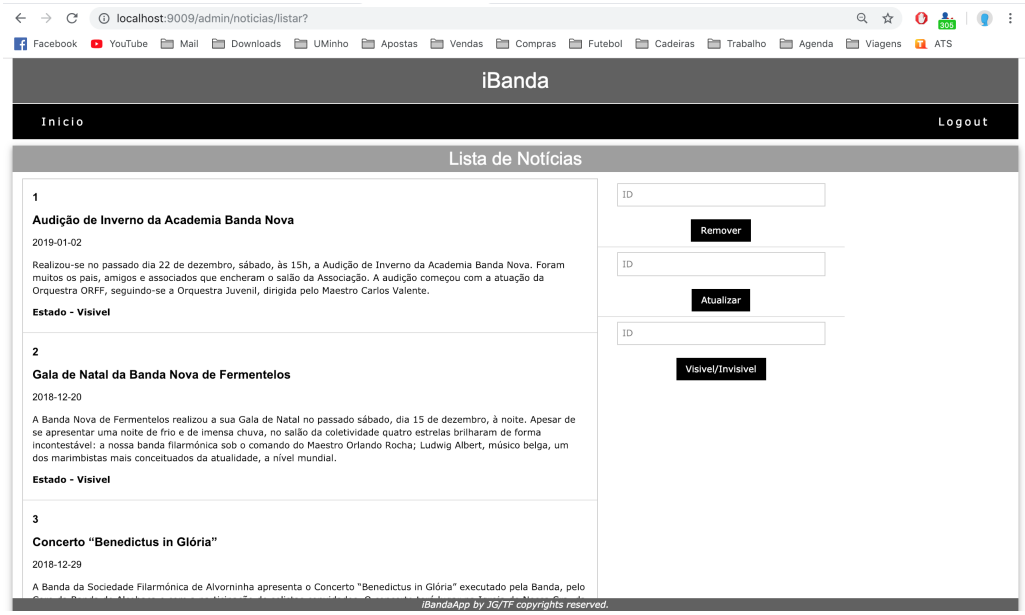

Figura 3.4: **Administrador** - Listagem de Noticias.

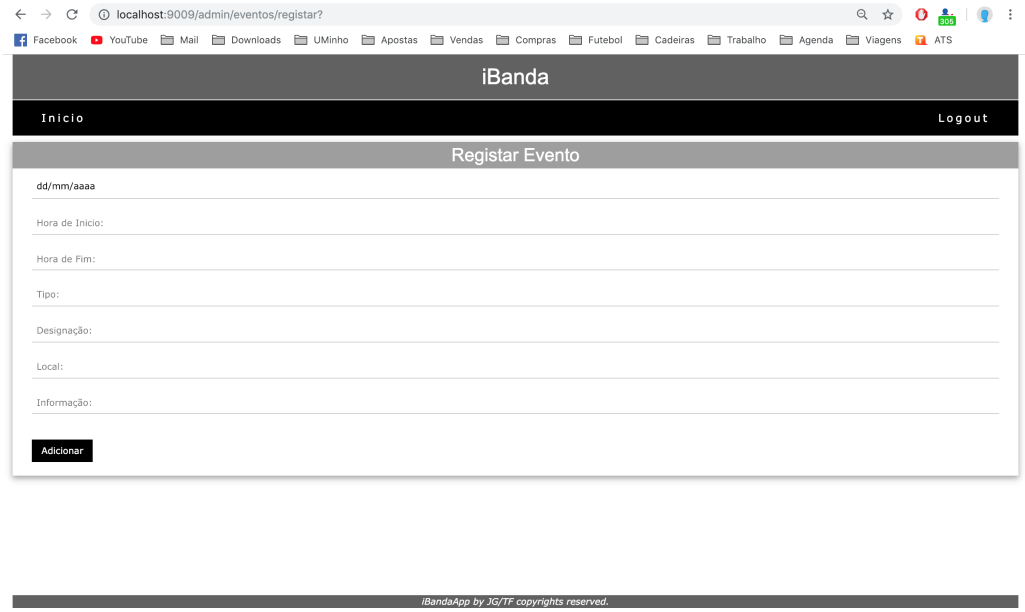

Figura 3.5: **Administrador** - Registo de Eventos.

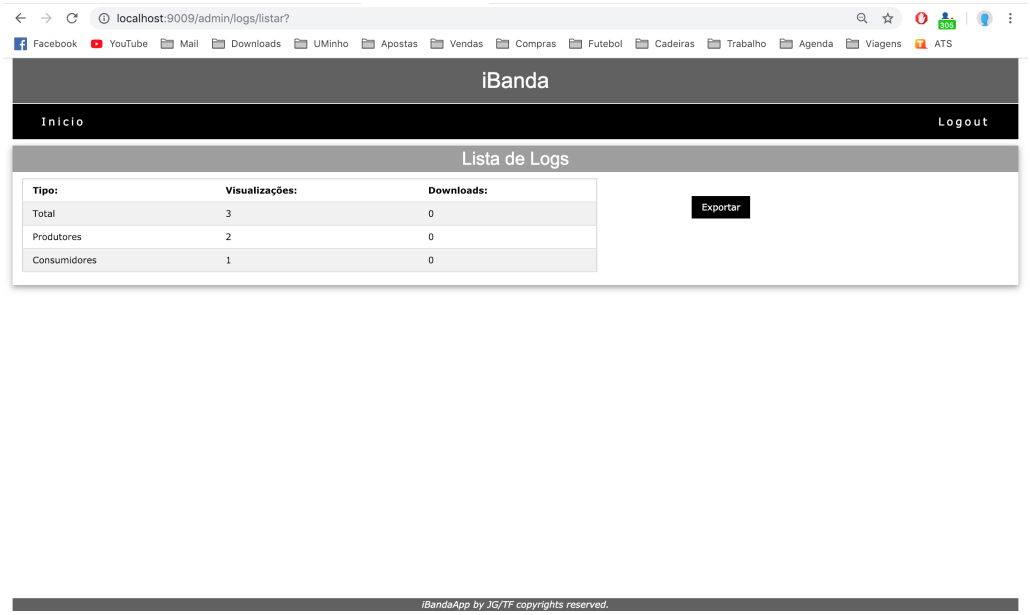

Figura 3.6: **Administrador** - Listagem das estatisticas da utilização da aplicação.

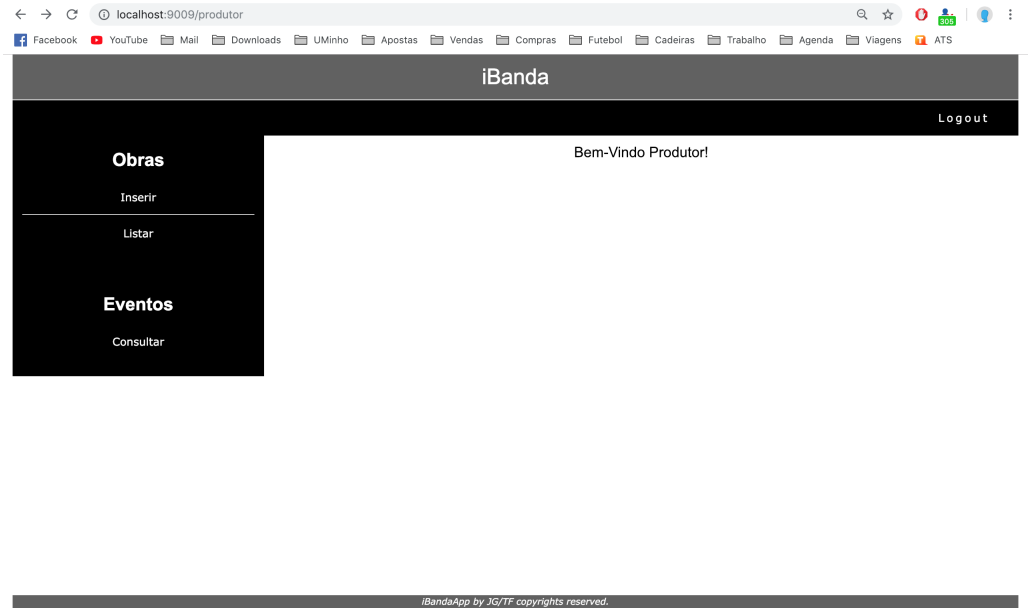

Figura 3.7: **Produtor** - Página Inicial.

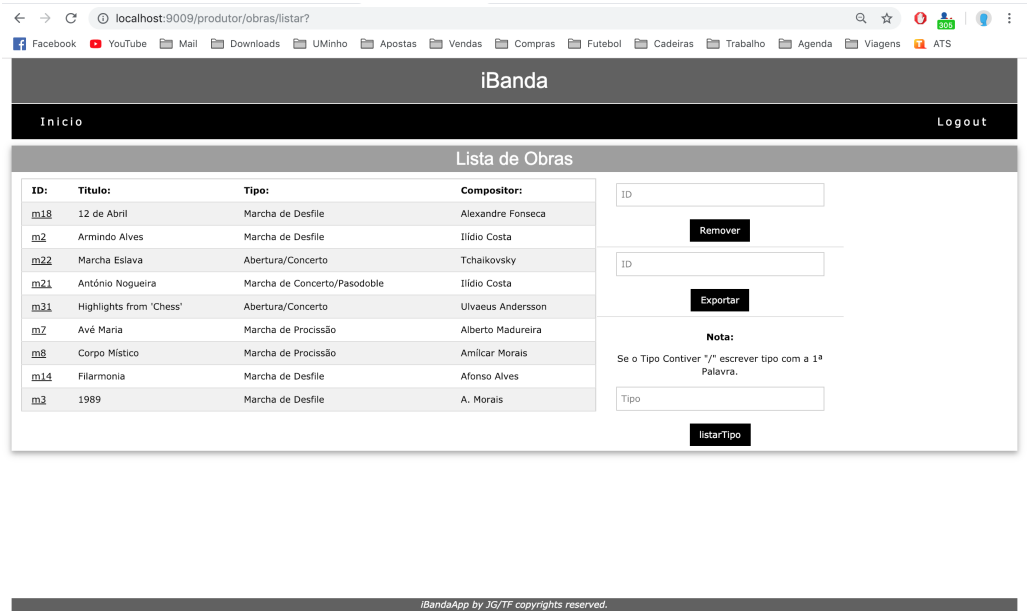

Figura 3.8: **Produtor** - Página de manipulação das obras.

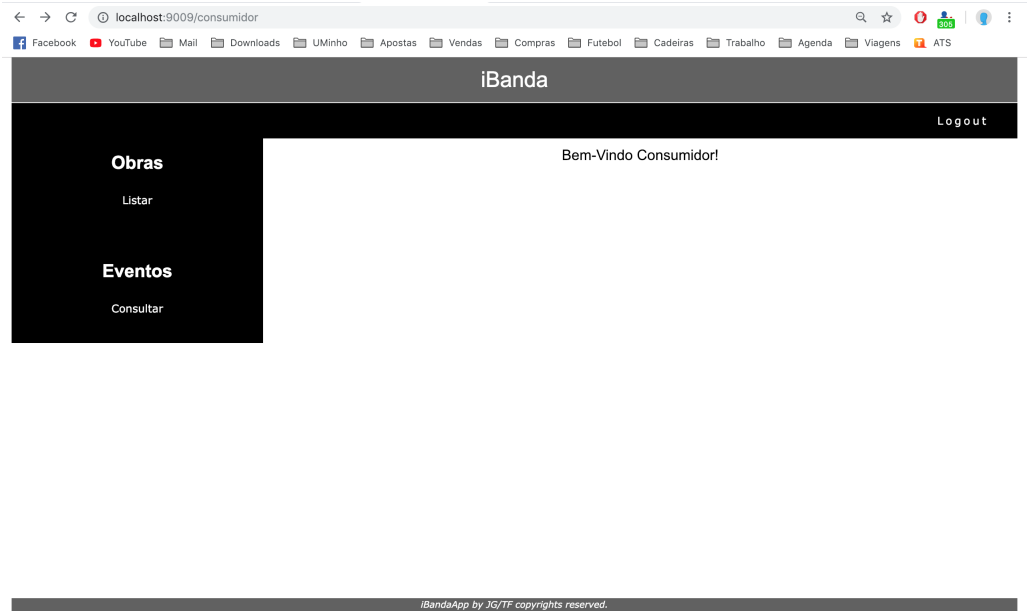

Figura 3.9: **Consumidor** - Página Inicial.

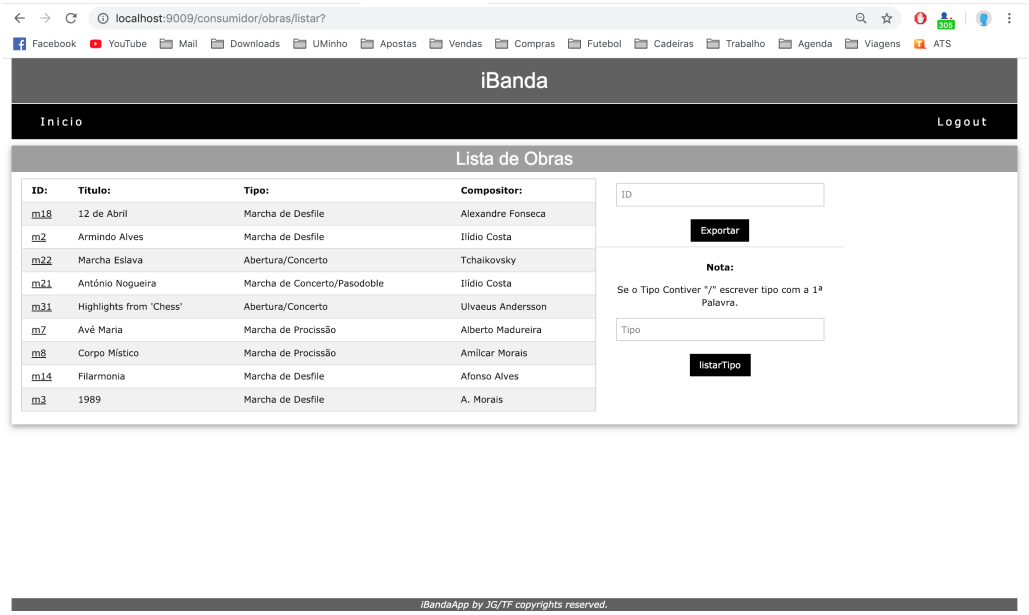

Figura 3.10: **Consumidor** - Listagem de obras.

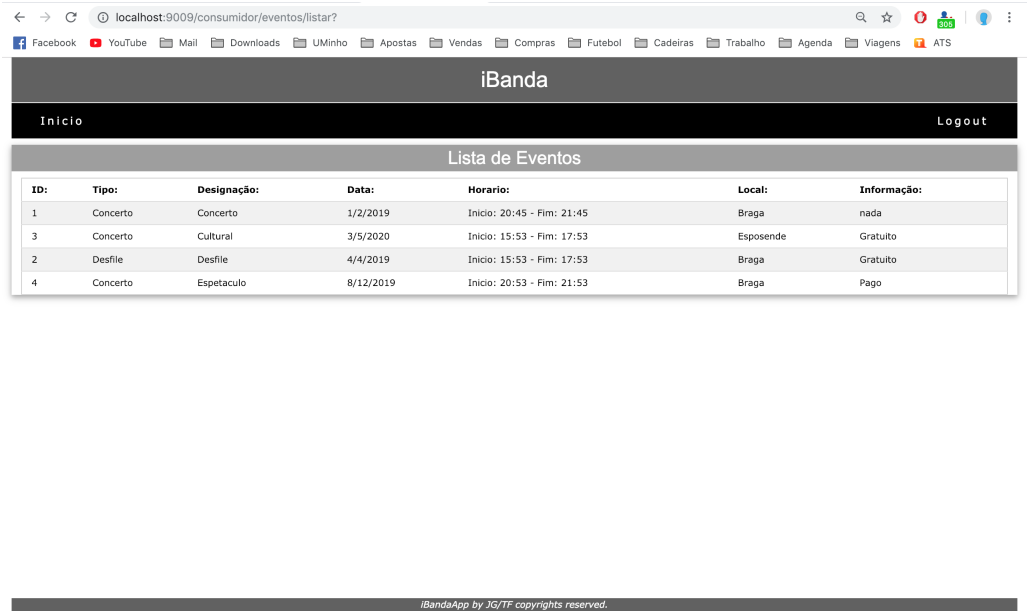

Figura 3.11: **Consumidor** - Listagem de Eventos.

Capítulo 4

Conclusões

Dado por terminado o trabalho efetuado, podemos constatar que a aplicação desenvolvida cumpre com todos os requisitos propostos. Apesar de pequenas arestas que precisam de ser limadas para libertar o projecto no mercado, podemos afirmar que a aplicação é fiável, robusta e fluida.

As principais dificuldades encontradas ao longo do desenvolvimento do projecto, foram em primeiro lugar, a gestão do tempo face ao elevado numero de tarefas que eram preciso ser implementadas. Em segundo lugar, a necessidade de autenticação no sistema estabeleceu-se como outra barreira a ultrapassar. Por último, a manipulação dos ficheiros *ZIP*, tanto para inserir como para exportar, assumiu-se como uma grande barreira a superar com o auxilio de ferramentas que não tinham sido utilizadas até então.

Como trabalho futuro poderíamos melhorar a interface, uma vez que utilizamos uma ferramenta simples, e que para determinados usos e utilizadores pode não ser apelativa o suficiente para aumentar exponencialmente o seu uso.

Em suma, este projecto contém todos as ferramentas e funcionalidades necessárias e bem implementadas para satisfazer utilizadores reais.