



Departamento de Engenharia de Electrónica e Telecomunicações e de
Computadores

49475 : João Nunes (A49475@alunos.isel.pt)

49424 : João Ramos (A49424@alunos.isel.pt)

Relatório para a Unidade Curricular de Introdução à Programação na Web da
Licenciatura em Engenharia Informática e de Computadores

Professor : Doutor Luís Falcão

Índice

1	Introdução	3
2	Elastic Search	4
2.1	Design do armazenamento de dados	4
2.2	Mapeamento entre Elastic Search e o modelo de aplicação	5
3	Documentação da API	7
3.1	Documentação Elastic Search	7
3.2	Documentação SECA	8.
3.3	Procedimento	9
4	Conclusão	10

1. Introdução

No âmbito de Introdução à Programação na Web iremos desenvolver uma aplicação SECA (Shows & Events Chelas Application). Esta aplicação permite aceder, através de uma interface web (hypermedia), a algumas das funcionalidades disponibilizadas pelo site (TicketMaster) (<https://developer.ticketmaster.com/>), fazendo uso da sua API Web (<https://developer.ticketmaster.com/products-and-docs/apis/getting-started/>). Para ter acesso à API, utiliza-se uma chave da API obtida no TicketMaster.

Para a realização desta aplicação foram necessárias 3 etapas de desenvolvimento: uma primeira parte onde foram implementadas as rotas principais da aplicação com respostas em JSON; uma segunda parte já com uma interface Web e com o armazenamento da memória na base de dados do Elasticsearch e, por fim, uma terceira fase onde o objetivo principal foi adicionar a autenticação do cliente à aplicação Web. Assim sendo, o desenvolvimento desta aplicação foi feito de forma incremental, ou seja, a cada fase foram-se adicionando “peças” à aplicação até ao resultado final.

A estrutura da aplicação encontra-se dividida em dois componentes, sendo eles o cliente e o servidor. Na componente Cliente a aplicação está estruturada em várias páginas: Register, Login, Home, Get Popular Events, Search an Event, GroupDetails e EventsDetails.

Para cada uma delas estão definidas no lado do servidor as rotas correspondentes a cada uma delas, bem como aquelas que apesar de não aparecerem no lado do cliente auxiliam nas restantes operações pretendidas pelo projeto (ex: editGroup, deleteGroup, addEvent e removeEvent).

Na componente Servidor estão presentes as views responsáveis por apresentar um aspeto mais apelativo para cada página web recorrendo a ferramentas como HandleBars, CSS e HTML.

2. Elastic Search

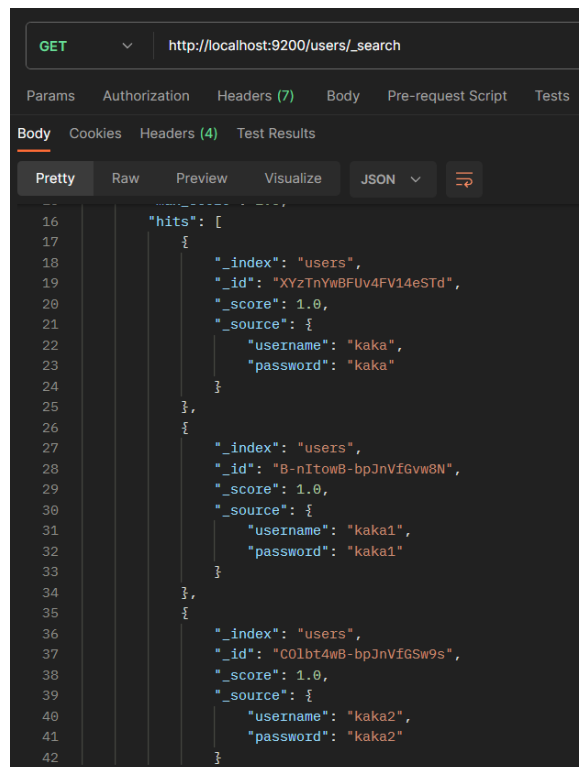
2.1 Design do armazenamento de dados

Nesta fase do trabalho, a ferramenta utilizada para armazenar os dados provenientes de cada grupo e cada utilizador foi o Elastic Search, uma base de dados local que funciona com base em índices e documentos.

Um índice é um conjunto de documentos que estão relacionados entre si e um documento é a unidade básica de informação codificada em JSON. Esta informação pode ser algo como strings, números ou datas. Cada índice e documento possui um id único, sendo que o documento possui uma propriedade *_source* onde se encontra a informação que se pretende armazenar.

Para uma melhor compreensão desta ferramenta, tomamos como exemplo o desenvolvimento da nossa aplicação onde, numa versão inicial, foi utilizado o *Postman*. De modo a verificar o comportamento e o retorno destas funcionalidades foram criados dois índices para os componentes principais da SECA: um para os utilizadores e outro para os grupos. De seguida, foram adicionados novos pedidos para cada uma das situações possíveis: criar, apagar, obter e editar um documento, havendo mais um pedido para obter todos os documentos criados. Cada um destes corresponde a um pedido equivalente para os grupos, já que cada documento possui a informação referente ao grupo em questão.

Como exemplo, segue-se a imagem onde são evidenciadas as propriedades do documento, no caso em que estamos perante um pedido de *GET document* para obter todos os *users* existentes.



```
16      "hits": [
17        {
18          "_index": "users",
19          "_id": "XYzTnYw8Fuv4FV14eSTd",
20          "_score": 1.0,
21          "_source": {
22            "username": "kaka",
23            "password": "kaka"
24          }
25        },
26        {
27          "_index": "users",
28          "_id": "B-nItowB-bpJnVfGw8N",
29          "_score": 1.0,
30          "_source": {
31            "username": "kaka1",
32            "password": "kaka1"
33          }
34        },
35        {
36          "_index": "users",
37          "_id": "C0lbt4w8-bpJnVfGSw9s",
38          "_score": 1.0,
39          "_source": {
40            "username": "kaka2",
41            "password": "kaka2"
42          }
43        }
44      ]
45    }
46  }
```

Como é possível observar, cada documento possui três propriedades fundamentais: o índice a que pertence (*users*), o seu id único e a sua *_source*, que contém toda a informação necessária de um *user*.

2.2 Mapeamento entre Elastic Search e o modelo de aplicação

Com o intuito de armazenar data na base de dados do Elastic Search, é necessário criar módulos capazes de interagir com o mesmo e de replicar o mesmo comportamento que os pedidos no Postman. Para tal, é utilizado o módulo *fetch*.

Como consequência destas novas implementações, os módulos em memória da data dos grupos e utilizadores são inutilizados, sendo apenas úteis na realização de testes.

Resumidamente, para cada operação a ser realizada é necessário fazer um *fetch* ao URI respetivo, de modo a obter uma resposta com a informação a que queremos aceder. Para cada pedido, é necessário referir o índice do documento, o método do pedido (GET, PUT, DELETE ou POST) e, caso assim o exigir, o id do grupo ou utilizador respetivo.

Neste exemplo para deletar um grupo, após obter o grupo pretendido iremos colocar o URI pretendido para deletar um documento no Elastic Search sendo feita através de “*URI_MANAGER.delete(id)*” em que *URI_MANAGER* é o index do documento, neste caso chamado de “groups”. A função delete cria o suposto URI para fazer acesso ao Elastic Search.

```
async function deleteGroup(user, id){
  let deleted = {}

  let group = await getGroup(user, id)

  deleted = {
    id: group.id,
    name: group.name,
    description: group.description,
    events: group.events
  }

  await del(URI_MANAGER.delete(id))

  return deleted
}
```

```
export const URI_PREFIX='http://localhost:9200/'

export default async function(index) {

  await put(`${URI_PREFIX}${index}`)

  return {
    getAll: () => `${URI_PREFIX}${index}/_search`,
    get: (id) => `${URI_PREFIX}${index}/_doc/${id}`,
    create: () => `${URI_PREFIX}${index}/_doc/?refresh=wait_for`,
    update: (id) => `${URI_PREFIX}${index}/_doc/${id}?refresh=wait_for`,
    delete: (id) => `${URI_PREFIX}${index}/_doc/${id}?refresh=wait_for`,
  }
}
```

Após ter o URI completo através da função “del” irá colocar o método pretendido e irá realizar fetch, neste caso deletando o grupo chamado na função inicial.

```
export async function get(uri) {
  return fetchInternal(uri)
}

export async function del(uri) {
  return fetchInternal(uri, { method: "DELETE" })
}

export async function put(uri, body) {
  return fetchInternal(uri, { method: "PUT" }, body)
}

export async function post(uri, body) {
  return fetchInternal(uri, { method: "POST" }, body)
}
```

```
async function fetchInternal(uri, options = { }, body = undefined) {
  if(body) {
    options.headers = {
      'Content-Type': 'application/json'
    }
    options.body = JSON.stringify(body)
  }

  const rsp = await fetch(uri, options)
  const obj = await rsp.json()
  return obj
}
```

3. Documentação da API

3.1 Documentação Elastic Search

Path base para os seguintes pedidos (de acordo com a configuração do Elastic Search):

http://localhost:9200/

Descrição	Pedido	Path	Query?	Body?
Cria um índice para grupos	PUT	groups	✗	✗
Elimina o índice de grupos	DEL	groups	✗	✗
Cria um grupo	POST	groups/_doc	✓	✓
Retorna o grupo de um utilizador	GET	groups/_doc/<group token>	✗	✗
Elimina os grupos de um utilizador	DEL	groups/_doc/<group token>	✓	✗
Edita um grupo	PUT	groups/_doc/<group token>	✓	✓
Retorna o documento dos grupos	GET	groups/_search	✗	✗
Cria um índice para utilizadores	PUT	users	✗	✗
Elimina o índice de utilizadores	DEL	users	✗	✗
Cria um utilizador	POST	users/_doc	✓	✓
Retorna um utilizador	GET	users/_doc/<user token>	✗	✗
Elimina um utilizador	DEL	users/_doc/<user token>	✓	✗
Retorna o documento dos utilizadores	GET	users/_search	✗	✗

Path	Query	Body
groups/_doc	refresh=wait_for	"name": String, "description": String or NULL
groups/_doc/<group token>	refresh=wait_for	-----
groups/_doc/<group token>	refresh=wait_for	"userId": user._id "name": String or NULL, "description": String or NULL, "events": Array
users/_doc	refresh=wait_for	"username": String, "password": String
users/_doc/<user token>	refresh=wait_for	-----

3.2 Documentação SECA

Path base para os seguintes pedidos (de acordo com a const *PORT* no ficheiro *seca-server-passport.mjs*): <http://localhost:6969/site/>

Todas as rotas que tiverem */public* são rotas publicas o que significa que não é necessário estar autenticado, já as rotas */private* são paginas que são necessárias estar autenticado.

Descrição	Pedido	Path	Query?	Body?	Auth?
Cria um utilizador	POST	private/HomeUser	✗	✓	✗
Cria um grupo	POST	private/CreateGroup	✗	✓	✓
Atualiza um grupo	PUT	private/editingGroup/:id	✗	✓	✓
Retorna os grupos de um utilizador (mostrado no home do user)	GET	private/Userhome	✗	✗	✓
Elimina um grupo de um utilizador	DEL	private/delete/:id	✗	✗	✓
Retorna os detalhes de um grupo	GET	private/oneGroup/:id	✗	✓	✓
Adiciona um evento a um grupo	POST	private/addEvent	✗	✓	✓
Remove um evento de um grupo	DEL	private/removeEvent/:id	✗	✗	✓
Retorna os detalhes de um evento	GET	public/oneEvent/:id	✗	✗	✗
Retorna os eventos resultantes de uma pesquisa	GET	public/searchEvents	✓	✗	✗
Retorna os eventos populares	GET	public/popularEvents	✓	✗	✗

4

Path	Query	Body
private/HomeUser	-----	"username": String, "password": String
private/CreateGroup	-----	"name": String, "description": String or NULL
private/editingGroup/:id	-----	"name": String or NULL, "description": String or NULL
private/addEvent	-----	" GroupId ": String, " eventId ": String
public/searchEvents	eventName = String limit = Integer page = Integer	-----
public/popularEvents	limit = Integer page = Integer	-----

3.3 Procedimento

Para usar a aplicação é necessário:

1. Clonar o repositório para uma pasta à escolha usando o Visual Studio Code.
2. Instalar, localmente ou globalmente, os seguintes pacotes, caso ainda não estejam instalados: *passport*, *express*, *cors*, *cookie-parser*, *url* e *path*, *expressSession* usando o comando *npm install <nome do pacote>*.
3. Instalar o Elastic Search (<https://www.elastic.co/pt/downloads/elasticsearch>), seguindo as instruções do instalador.
4. Abrir o ficheiro de configuração do Elastic Search localizado na pasta *config* da instalação, modificar o valor de *xpack.security.enabled* para *false* e salvar o ficheiro.
5. Executar o ficheiro batch *elasticsearch* na pasta *bin*.

Deste modo, estão feitas as preparações para executar a aplicação e os seus testes.

Para executar a aplicação principal e os seus testes basta abrir um terminal e executar *npm run start-pass* para correr o servidor e *npm run test* que irá executar os testes.

4. Conclusão

A divisão da aplicação em fases permitiu a testagem de módulos ao longo do desenvolvimento de modo a evitar erros no futuro, visto que a aplicação está hierarquicamente construída. Além disto, a modularidade permita a fácil adaptação de código para utilizar outros APIs, usar outras bases de dados orientado a documentos e ainda a implementação de outros métodos caso seja necessário.

5. WebGrafia

- Página de Github da turma: <https://github.com/isel-leic-ipw/2324i-IPW-LEIC31D>
- Documentação de HTTP: <https://developer.mozilla.org/en-US/docs/Web/HTTP>
- Documentação do módulo Mocha: <https://mochajs.org/>
- Documentação do módulo Express: <https://expressjs.com/en/4x/api.html>
- Documentação do HandleBars <https://handlebarsjs.com/guide/>
- Documentação de HTML: <https://developer.mozilla.org/en-US/docs/Web/HTML>
- Documentação de CSS: <https://developer.mozilla.org/en-US/docs/Web/CSS>