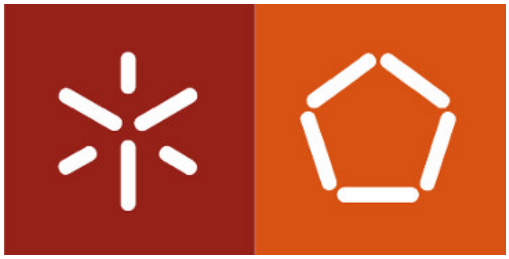

Universidade do Minho

Mestrado integrado em Engenharia Informática

Trabalho Prático

Cândido Filipe Lima do Vale, PG42816

Diogo Pinto Ribeiro, A84442

Luís Paulo Freitas da Cunha, A84244

Desenvolvimento de Aplicações Web

4º Ano, 1º Semestre

Departamento de Informática

7 de fevereiro de 2021

Índice

Lista de Figuras

1 Introdução

Neste relatório iremos apresentar o projeto desenvolvido no âmbito da unidade curricular de Desenvolvimento de Aplicações Web. O projeto consiste numa plataforma de partilha de recursos educativos que permite o registo de utilizadores, a criação de posts acerca de recursos bem como a publicação e descarga destes recursos. Os recursos possuem diversas propriedades como a sua visibilidade ou uma taxonomia classificativa feita com a utilização de *hashtags*. Para o desenvolvimento desta plataforma foram utilizados principalmente **Node.js** e **MongoDB** a nível dos servidores e **JavaScript** do lado do cliente.

2 Arquitetura

A aplicação por nós desenvolvida consiste nos três componentes a seguir apresentados. A razão pela qual efetuámos esta divisão passou pelo facto de uma **arquitetura de micro-serviços** ser um padrão que nós considerámos adequados à problemática em causa e permitir facilmente a adição de componentes novas, bem como facilitar na identificação de eventuais problemas.

2.1 API Server

O **API Server** é o componente que efetua todas as transações sobre dados armazenados tais como adições ou remoções de recursos ou posts. O API Server pode ser contactado de duas formas: diretamente através da API exposta. ou através do Interface Server.

Todos os pedidos que feitos ao API Server necessitam de possuir um **token válido** para poderem ser respondidos sendo que para esse efeito usamos **JSON Web Tokens**. Além da verificação do token é feita uma verificação adicional para determinadas rotas, onde um utilizador com **privilégios de administrador** possui acesso à operação. Um exemplo disso são os recursos, onde um administrador tem permissão para qualquer uma das operações **CRUD**.

2.1.1 Modelos

Os modelos existentes no API Server passam pelos modelos do **Utilizador**, **Recurso**, **Tipo**, **Post** e **Notícia**. Para o **Utilizador** guardámos os campos **username**,

nome, **email**, **filicação**, **curso** ou **departamento**, **instituição**, **admin** (true ou false), **dataRegisto**, **ultimoAcesso**, **hashedPassword**, **starred** (array de identificadores de recursos que o utilizador atribuiu uma estrela) e **salt**. Para o **Recurso** são armazenados os valores **tipo**, **título**, **subtítulo**, **dataCriação**, **dataRegisto**, **visibilidade**, **nome**, **mimetype**, **autor**, **stars** e **tags**. Para o **Tipo** apenas guardámos o campo **id** que contém o nome do tipo e o campo **recursos** que representa o número de recursos do tipo. Os Posts são constituídos pelos campos **utilizador**, **texto**, **recurso**, **data** e **comentários** (que por sua vez é um array de objetos com os campos utilizador, data e texto). Por fim, as **Notícias** possuem os campos **utilizador**, **texto**, **recurso** e **data**.

2.1.2 Rotas

```
GET /utilizadores
GET /utilizadores/:id
GET /utilizadores/detalhes/:id
PUT /utilizadores/:id
DELETE /utilizadores/:id

GET /tipos
GET /tipos/top/:n
GET /tipos/:id
POST /tipos
PUT /tipos/:id
DELETE /tipos/:id

GET /recursos
GET /recursos/novo
GET /recursos/estatisticas
GET /recursos/exportar
GET /recursos/:id
GET /recursos/zip/:id
POST /recursos
POST /recursos/importar
PUT /recursos/:id
DELETE /recursos/:id
PUT /recursos/star/:id

GET /posts
GET /posts/:id
POST /posts
PUT /posts/:id
DELETE /posts/:id
POST /posts/comentarios/:id
DELETE /posts/comentarios/:id

GET /noticias
GET /noticias/:id
POST /noticias
PUT /noticias/:id
DELETE /noticias/:id
```

2.2 Interface Server

O **Interface Server** é o componente que tem como principal tarefa efetuar pedidos ao API Server e retornar como resposta os dados obtidos apresentados graficamente. Este servidor utiliza o *view engine* **Pug** para a estruturação das páginas, em conjunto com **CSS** que as torna *responsive*. As páginas possuem também **javascript** e operações **AJAX**, que modificam a página sem necessitar de a recarregar.

As principais são páginas são: a página de *login*, a página principal (*feed*), a página de apresentação dos recursos, a página individual de um recurso e a página de admin.

Na página de *login* é possível criar conta, podendo ser criada a partir do *google*, *facebook* ou nenhum dos dois. É possível também recuperar a palavra-passe e efetuar o *login*.

Na página principal são mostrados os *posts* feitos a recursos visíveis ao utilizador, as notificações relativas à adição destes recursos e as notícias criadas pelos administradores. É possível também navegar nos recursos por tipos, ver os recursos adicionados pelo utilizador e fazer o *upload* de um recurso.

Na página dos recursos são apresentados os recursos visíveis ao utilizador, podendo visualizar apenas os recursos de um certo tipo ou os recursos adicionados pelo utilizador. É possível também adicionar um novo recurso.

Na página individual de um recurso é mostrado o conteúdo do recurso (para os formato compatíveis), é mostrada a meta-informação são disponibilizadas as operações de criação de um post relativo ao recurso, de descarregamento e, caso o utilizador tenha permissões, de edição e remoção do recurso.

Por fim, a página de administrador permite, a um utilizador com permissões de *admin*, importar e exportar recursos, ver as estatísticas, adicionar notícias, ver as suas notícias publicadas e apagá-las.

2.3 Auth Server

O **Auth Server** é o componente responsável pelo registo, autenticação dos utilizadores na aplicação e recuperação de password em caso de esquecimento. Será responsável, no processo de autenticação, por verificar as credenciais introduzidas pelos utilizadores ou os tokens recebidos de mecanismos de OAuth e emitir os tokens JWT que os utilizadores poderão utilizar para aceder à aplicação.

2.3.1 Modelos

No Auth Server apenas utilizamos um modelo do **Utilizador** que é idêntico ao utilizado no API Server.

2.3.2 Rotas

```
POST /utilizadores
POST /utilizadores/login
POST /utilizadores/recuperaPassword
POST /utilizadores/redefinePassword
POST /utilizadores/googleAuth
POST /utilizadores/facebookAuth
```

3 Recursos

Os **Recursos** são um dos componentes centrais da nossa plataforma. De momento os ficheiros que podem ser submetidos incluem os formatos **.txt**, **.pdf**, **.zip**, **.doc**, **.docx**, **.xls** e **.xlsx**.

Após estarem no sistema os recursos com a visibilidade definida como **Público** poderão ser vistos por todos os utilizadores do sistema, enquanto que aqueles que possuem a visibilidade a **Privado** apenas podem ser visualizados pelos respetivo autor. O autor é também quem pode editar ou eliminar o respetivo recurso. A única exceção a estas regras são os utilizadores com privilégios de administrador que podem aceder a qualquer recurso, Público ou Privado, podendo também editar ou remover os mesmos.

3.1 Estrutura

Tal como sugerido no enunciado foram utilizadas as noções de **SIP**, **DIP** e **AIP**. Por motivos de compatibilidade e por considerarmos que seria a opção mais lógica no nosso contexto considerámos que os SIP e DIP possuem o mesmo formato. Este formato é baseado no *BagIt File Packaging Format* e tem a seguinte estrutura:

```
recurso/
|   manifest-sha256.txt
|    (49afbd86a1ca9f34b677a3f09655eae9 data/exemplo.pdf)
|   bagit.txt
|    (BagIt-version: 1.0)
|    (Tag-File-Character-Encoding: UTF-8)
\--- data/
     |   exemplo.pdf
```

Relativamente a um Recurso que se encontre armazenado a sua estrutura é relativamente simples. Toda a sua informação é guardada no modelo apresentado na secção anterior, e o ficheiro contido no SIP é guardado no servidor, mais concretamente dentro de uma árvore contida na pasta recursos. Simulando o cenário em que existe um elevado número de recursos pode-se levantar a questão de atingirmos o limite de ficheiros armazenados numa única pasta. Essa problemática motivou-nos a arranjar a seguinte solução: após ser inserido na base de dados, o **identificador gerado pelo MongoDB** é utilizador para construir o caminho na árvore. Isto é feito **partindo o identificador em 4 partes iguais** com 6 carateres cada. A divisão deste identificador pode ser mais granular de modo a suportar um número superior de inserções, mas no nosso caso a partição utilizada é suficiente. Por exemplo, o recurso cujo identificador seja *601ac39fbdd51b0804bab574*, será armazenado no caminho **recursos/601ac3/9fbdd5/1b0804/bab574/**, sendo criado no caso de não existir. Esta implementação gera uma árvore que resolve a limitação referida.

4 Funcionalidades

Após perceber-mos melhor quais as componentes que constituem o nosso sistema e como este funciona, é também necessário perceber o que este se compromete a realizar. As principais funcionalidades do nosso sistema passam por:

4.1 Publicação, Visualização e Descarga de Recursos

A funcionalidade mais básica do nosso sistema consiste na Publicação, Visualização e Descarga de Recursos educativos. A Publicação e Descarga de Recursos obedece o formato SIP e DIP descrito anteriormente, sendo que existem alguns detalhes adicionais. Recursos Públicos podem ser visualizados por qualquer utilizador, sendo que no caso

do Recurso ser Privado apenas o autor ou um administrador o poderá ver. Relativamente a operações *CRUD*, estas são também exclusivas ao autor do Recurso ou a um administrador.

A visualização de um Recurso no browser é possível para determinados tipos nomeadamente documentos pdf e txt, dado que são suportados pelo próprio browser. A procura de Recursos teve uma atenção especial sendo possível listar Recursos com base no seu **autor**, nas suas **tags**, e no seu **Tipo**. No caso das tags, é possível um Recurso possuir múltiplas tags que o classificam, aparecendo na pesquisa por cada uma delas.

4.2 Estrelas

Uma das funcionalidades do nosso sistema passa pela possibilidade de atribuir **estrelas** a recursos. O principal objetivo é a implementar um mencanismo de avaliação dos recursos, onde à semelhança de plataformas como o **GitHub**, recursos com um maior número de estrelas podem ser relacionados com um conteúdo mais apelativo. É também possível remover estrelas atribuídas.

A informação é guardada em dois locais distintos: no recursos é guardado o número de estrelas atribuídas, e no utilizador é guardada a lista de recursos a que atribuiu estrelas.

4.3 Posts

É possível publicar um *post* relativo a um recurso, a utilizadores que consigam ver o recurso. Estes *posts* são apresentados no *feed* da página principal, apenas a utilizadores que consigam ver o recurso. Contêm uma hiperligação para o recurso, o texto do *post* e uma secção de comentários. Os *posts* e os comentários podem ser removidos por administradores ou pelo utilizador que os publicou.

4.4 Notícias

As notícias são apresentadas também na página inicial, e ao contrário dos *posts*, não possuem uma secção de comentários. Estas são publicadas por administradores (na página de administração) ou são geradas automaticamente quando um utilizador faz o *upload* de um recurso. Neste último caso, também possuem uma hiperligação para o recurso. As notícias podem ser removidas por administradores ou pelo utilizador a quem o *upload* diz respeito.

4.5 Privilégios de Administração

Utilizadores que possuam privilégios de administração possuem um conjunto de funcionalidades acrescidas face aos restantes utilizadores. Estas passam pela possibilidade de ver Recursos Privados e de poder editar ou eliminar qualquer recurso existente. Administradores têm ainda acesso a uma página de administração no qual podem inserir Notícias no sistema, bem como consultar estatísticas e importar ou exportar Recursos.

4.6 Estatísticas de Utilização

Uma funcionalidade que achámos que acrescentaria valor ao sistema é a visualização de estatísticas. Estas permitem-nos perceber melhor a utilização do sistema e a evolução do mesmo. De momento as estatísticas disponibilizados passam pelo **número de Recursos por Tipo**, **número total de Utilizadores** e **número de Docentes/Alunos**, **total de Recursos**, **número de Recursos inseridos nos últimos 7 dias**, e, pelo **número de utilizadores que efetuaram login pela última vez nos últimos 7 dias**.

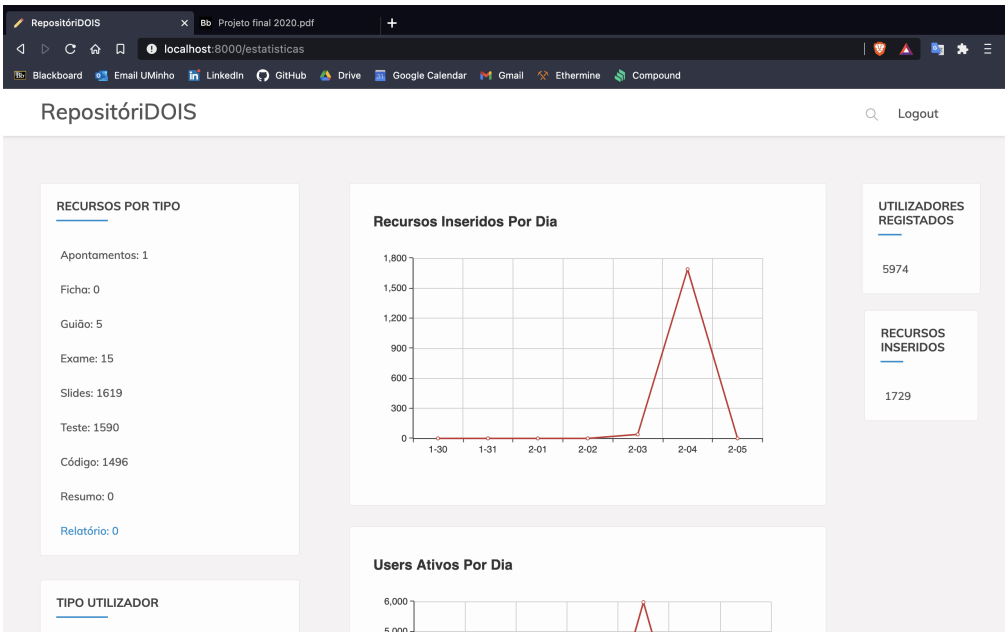

Figura 1: Página Estatísticas

Estas estatísticas são um processo iterativo, podendo numa versão futura serem adicionadas novas estatísticas mais relevantes ou serem recolhidos mais dados para criar novas estatísticas.

4.7 Importação e Exportação

Um dos requesitos propostos foi a possibilidade de exportar e importar Recursos, sendo obrigatório a possibilidade de importar o resultado de uma exportação. No nosso caso, decidimos aproveitar o formato dos SIP/DIP e efetuar a exportação agregando todos os Recursos no formato SIP/DIP dentro de um ficheiro **zip**. Esta estrutura é de fácil obtenção e relativamente eficiente. Para a importação é esperado um ficheiro zip no formato descrito anteriormente. Após o API Server receber o zip com os Recursos a importar, efetua a operação de adição do Recurso na pltaforma individualmente, retornando o número de recursos inseridos no sistema, ou no caso de todos existirem, retorna uma mensagem específica com o erro em questão.

4.8 Registo e Autenticação de Utilizadores

Outra das funcionalidades mais básicas necessária ao correto funcionamento do sistema é a funcionalidade de registo e posterior autenticação de utilizadores.

Desta forma, implementamos na nossa aplicação um mecanismo de registo que permite aos utilizadores criar uma conta no nosso sistema sendo que, posteriormente, podem autenticar-se e aceder à aplicação quer através de um login tradicional quer através de mecanismos OAuth como autenticação com conta Google ou Facebook.

Em ambos os cenários, caso se verifique que o utilizador está registado no nosso sistema e a sua password ou token são válidos será enviado ao utilizador um JWT que depois será utilizado em todos os pedidos para o Interface e API Servers de modo a permitir que o acesso do utilizador aos dados. Nesta implementação, decidimos utilizar JWT por ser Stateless, o que significa que não precisámos de armazenar qualquer informação acerca dos tokens nos nossos servidores e portanto, permitirá escalar por exemplo, para múltiplos API Servers mais facilmente do que se estivéssemos a armazenar sessões.

No caso de um utilizador tentar efetuar Login com um dos mecanismos OAuth mas não tiver ainda conta, tentámos tornar o processo de registo um pouco mais ágil recolhendo logo o email e nome associados e apresentando um formulário de registo mais simplificado. Neste caso, optámos por esta metodologia pois o registo na nossa aplicação requer o preenchimento de dados que consideramos que não faria grande sentido serem gerados de forma automática ou com valores default.

Por fim, implementamos também um mecanismo que irá permitir o Logout dos

utilizadores e que irá consistir, para além de destruir o cookie com o token na máquina do cliente, em adicioná-lo a uma blacklist que é mantida no API Server onde existe também uma função a correr de limpeza a correr periodicamente de modo a limpar os tokens que estavam na blacklist mas já expiraram. Neste caso optámos por fazer a persistência dos tokens na blacklist num dicionário local por simplicidade de implementação. A forma ideal de implementar esta blacklist de tokens seria recorrendo a uma base de dados em memória tipo o **Redis**. Também consideramos fazer esta persistência utilizando o Mongo mas achamos que tal não seria adequado uma vez que uma das principais features dos JWT é ser stateless e ao armazenar os tokens na blacklist no Mongo estaríamos a ir contra esse princípio.

4.9 Recuperação de Password

A funcionalidade de recuperação de passwords é uma funcionalidade básica e que está presente em praticamente todas as aplicações uma vez que é comum os utilizadores esquecerem-se das suas passwords. Desta forma, considerámos que seria importante ter esta funcionalidade na nossa aplicação de modo a que um utilizador que se esquecesse da sua password não ficasse sem acesso à mesma ou tivesse que criar uma nova conta.

A estratégia adoptada na implementação desta funcionalidade passa por disponibilizar uma página onde um utilizador pode requisitar que lhe seja enviado para o email de registo um link que lhe irá permitir fazer o reset da password (note-se que este link apenas pode ser utilizado para fazer o reset uma vez).

5 Correr a Plataforma

De modo a sermos capazes de correr a plataforma é necessário ter o **MongoDB** e o **node** instalado. Após a instalação dos requisitos, basta correr em cada uma das pasta de cada serviço, **auth-server**, **api-server** e **interface-server** o comando **npm i** para instalar as depêndencias, e correr o comando **npm start** para lançar cada um dos serviços. Antes da plataforma estar pronta a correr é ainda necessário usar o **mongoimport** para importar os Tipos na nossa base de dados, usando o seguinte comando (com o ficheiro contido na pasta data):

```
mongoimport --db RepositoriDOIS --collection tipos
--file tipos.json --jsonArray
```

Ao importar os documentos com sucesso, passámos a ter um conjunto base de Tipos no sistema e a plataforma passa a estar operacional. É também fornecido um dataset com perto de 6000 utilizadores na mesma pasta que pode ser utilizado com o seguine comando:

```
mongoimport --db RepositoriDOIS --collection utilizadores
--file utilizadores.json --jsonArray
```

Existe também a possibilidade de converter um utilizador normal em administrador, sendo que para isso basta correr o seguinte comando no mongo, substituindo USER pelo *username*:

```
db.utilizadores.update({username: "USER"},{$set: {admin: true}})
```

Na página de administração é também possível importar um dataset de recursos disponibilizado na pasta data, sendo apenas selecionar o zip correspondente quando pedido.

6 Conclusões Finais

Após terminarmos a implementação da plataforma e realizarmos alguns testes efetuámos uma breve reflexão acerca do que foi realizado. Um dos principais pontos a que chegámos passou pela dificuldade em lidar com ficheiros ser um pouco mais elevada que o antecipado e requerer um cuidado especial. Percebemos também que existem alguns pontos no sistema que poderiam melhorar de modo a tornar o sistema mais consistente, nomeadamente no download de ficheiros, onde detetámos uma pequena inconsistência no interface-server que necessitaria de uma revisão numa iteração futura.

Na generalidade ficámos satisfeitos com os resultados obtidos, especialmente por termos conseguido construir uma plataforma completamente funcional que cumpre os objetivos por nós definidos e por termos adquirido novos conhecimentos pelo caminho.