Universidade do Minho
Escola de Engenharia

Sistema de Gestão de Recursos

Desenvolvimento de Aplicações Web

Trabalho realizado por:

Diogo Ferreira (PG42824)

Filipe Freitas (PG42828)

Fevereiro de 2021

Índice

Capítulo 1

Introdução

1.1 Contextualização

No âmbito da UC de Desenvolvimento de Aplicações Web foi proposta a realização de um trabalho prático, que consiste no desenvolvimento de uma aplicação Web para a gestão de recursos educativos. A referida aplicação deverá ter suporte para vários utilizadores, vários níveis de acesso e, para testes, deverá ser inicializada com algumas centenas de recursos.

1.2 Estrutura do relatório

Este relatório está dividido em seis partes:

- No primeiro é feita uma pequena introdução do trabalho e é descrita a estrutura do relatório.
- No segundo é feita uma *overview* da arquitetura da aplicação desenvolvida.
- No terceiro é feita uma análise à API de dados desenvolvida.
- No quarto é feita uma análise à aplicação de *frontend* desenvolvida.
- No quinto é descrito o processo de povoamento da aplicação.
- No sexto é feita uma conclusão sobre o trabalho desenvolvido.

Capítulo 2

Overview da aplicação desenvolvida

A aplicação desenvolvida encontra-se dividida em dois componentes:

- **API de dados** - servidor responsável por toda a gestão dos dados da aplicação;
- **Servidor de *Frontend*** - servidor responsável pela interface gráfica da aplicação;

Tal como o nome indica, a API de dados é responsável pelo armazenamento de toda a informação relativa aos recursos, com a notável excepção dos ficheiros que estão associados a cada recurso. Estes estão armazenados no *Frontend* por uma questão de *performance*. Além disso, este servidor funciona também como servidor de autenticação, ou seja, é responsável pelo registo, *login* e gestão dos utilizadores.

O servidor de *frontend* é responsável por toda a interface *Web*. É também neste servidor que é feito o armazenamento dos documentos associados a cada recurso, tal como foi referido anteriormente. O armazenamento dos documentos no servidor de *frontend* foi decidido de modo a não sobrecarregar a API de dados, e também para se evitar ter de se fazer mais do que um download quando se pretende fazer um download de ficheiros (download da API de dados para o servidor de *frontend*, e depois download de novo para o *browser* do cliente). Num trabalho futuro, tal como será mencionado à frente, esta gestão dos ficheiros deve ser isolada num servidor próprio. Devido à localização dos ficheiros, este servidor está também encarregue da produção e download dos *Bags* e da sua posterior ingestão.

Capítulo 3

API de Dados

3.1 Arquitetura do Servidor

O servidor da API de dados faz uso do *Express*, uma *framework* de desenvolvimento *Web* para *NodeJS*. Para a permanência de dados é utilizado o *MongoDB*, uma base de dados orientada ao documento. Recorremos também ao uso de *TypeScript* para permitir a *type-safety* da API de dados, para evitar erros de programação comuns e para facilmente detetarmos bugs no modelo de dados. O *TypeScript* assim foi extremamente útil para exigir a consistência dos dados da base de dados, e para não permitir modificações indevidas dos dados.

3.2 Modelo de Dados

Tal como foi referido, a API de dados faz uso do *MongoDB* para a permanência da informação. Assim sendo, apresentamos a seguir um modelo com todas as *collections* que foram criadas, bem como a estrutura dos documentos guardados por cada *collection*:

Quando é necessário aceder a dados de outra *collection*, é utilizado o plugin *mongoose-autofill* para automaticamente preencher os dados dos documentos guardados nas outras *collections*. Isto facilitou o desenvolvimento da API de dados, bem como o desenvolvimento do *frontend*, pois assim não é necessário fazer várias *queries* à API de dados para obter toda a informação necessária para o *frontend*.

Se a *array* de IDs a guardar for relativamente pequena, ou então, se mantiver relativamente estática durante toda a vida desse objeto, então os IDs dos objetos são guardados diretamente no documento parente. Se, como no caso dos comentários e das respostas a comentários, o número de subdocumentos puder aumentar rápidamente ao longo da vida do objeto, então optamos por guardar o ID do parente nos subdocumentos, e não guardar os IDs comentários e respostas diretamente no seu parente. Isto constitui uma boa prática de desenho de bases de dados, de acordo com o manual do *MongoDB* [1].

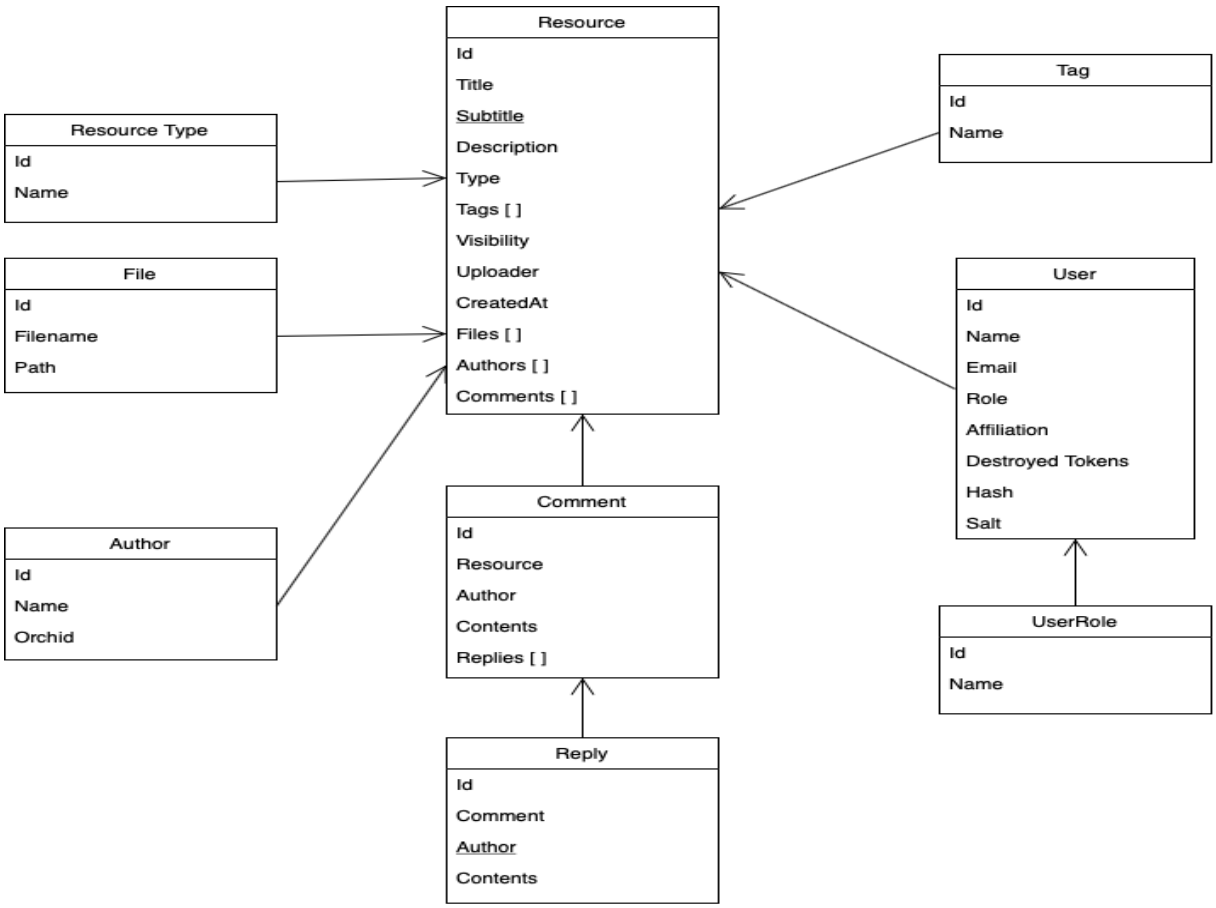

Figura 3.1: Modelos de dados do *MongoDB*

Para a modelação da base de dados dentro da aplicação, foi utilizada a biblioteca *typegoose*, que permite fazer a modelação dos *schemas* do *mongoose* com recurso ao sistema de classes do *TypeScript*, e mantendo também a *type-safety* dada por este. Isto fez com que fosse bastante mais fácil manter a consistência dos dados.

A modelação dos dados também foi feita de um modo mais genérico do que aquele que eventualmente foi necessário para implementar as funcionalidades pedidas: a criação de *schemas* distintos para autores, tags, e outros, fez com que seja possível relacionar dados que atualmente não estão a ser relacionados na interface gráfica (por exemplo, apesar de a interface gráfica não o permitir, é possível responder à questão "Quais são todos os recursos criados pelo autor X?". Deste modo, no futuro, é possível melhorar significativamente a interface gráfica sem ter de perder tempo a adicionar novas funcionalidades à API de dados; é também possível criar interfaces gráficas distintas (desde que seja resolvido o problema do armazenamento dos ficheiros antes). Devido à separação dos privilégios dos utilizadores na sua própria coleção, é possível, no futuro, implementar um sistema de permissões mais complexo (com várias *roles*, e com níveis de privilégios diferentes para cada uma das *roles*).

3.3 Autenticação

A API de dados também é responsável por efetuar a autenticação dos utilizadores. A autenticação está implementada nas rotas */auth*, sendo que, ao fazer login, é retornado um *token JWT*, que pode ser utilizado em pedidos subsequentes à API de dados.

Este token é armazenado pela interface gráfica e enviado para a API de dados sempre que for necessário.

Internamente, a autenticação está efetuada com recurso ao *passport*, *passport-local* e *passport-local-mongoose*. Esta última biblioteca faz, internamente, a cifragem da *password* do utilizador, guardado o resultado no campo *hash* (e o *salt* gerado no campo com o mesmo nome).

Atualmente, os tokens emitidos têm uma expiração de um dia, sendo que, ao final de um dia, é necessário voltar a efetuar *login* para se manter o acesso à base de dados. A duração pode ser ajustada no ficheiro de *environment* do servidor (disponível em *src/pre-start/env/*.env* na API de dados, e em *.env* no servidor de *frontend*; as durações devem ser ajustadas de uma forma igual em ambos os servidores).

3.4 Segurança das Rotas

A API de dados tem padrões de segurança implementados de acordo com os requisitos pedidos: para aceder a qualquer rota, pode ser exigida autenticação, mas também pode ser exigidos previlégios de edição ou visualização de certos recursos. Por exemplo, um utilizador não pode visualizar os recursos privados de outros utilizadores, excepto se o utilizador tiver atribuída a *role* de Administrador. A verificação de permissões é efetuada individualmente por cada rota. No entanto, foram criadas algumas funções de utilidade, que são chamadas pelas várias rotas. Estas funções (*checkResourceAccessPrivileges, checkResourceEditPrivileges* e *checkResourceCommentPrivileges*) estão disponíveis para visualização no ficheiro *src/routes/Resources.ts*, e estão documentadas de modo a ser explícito quais são as condições necessárias para estas autorizarem um acesso a um recurso.

Além destas funções, as restantes rotas que não as usam implementam os seus próprios mecanismos de verificação de permissões. Por exemplo, as rotas de procura de recursos apenas permitem a procura por recursos públicos, ou então por recursos que o utilizador seja dono. No caso de um administrador, não são impostas restrições às suas capacidades de pesquisa.

3.5 Instruções de execução

Para executar o servidor da API de dados, é ncessário:

1. Assegurar que temos a versão mais recente do *NodeJS* instalada (o servidor foi testado no *NodeJS* versão 15.3.0);

2. Efetuar *npm install* para instalar todos os pacotes necessários;

3. Ajustar a configuração de ambiente do servidor (em *src/pre-start/env/*.env*; escolher o ficheiro apropriado de acordo com a configuração que se quer correr);

4. Executar o MongoDB;

5. Executar o servidor:

    - Em modo de desenvolvimento - *npm run start:dev*;
    - Em modo de produção - *npm start*.

Nota importante: para a primeira execução, é necessário efetuar o povoamento da base de dados. O processo de povoamento é descrito no capítulo 5.

Capítulo 4

Aplicação de *Frontend*

4.1 Arquitetura do Servidor

Esta parte da aplicação foi desenvolvida utilizando *Express* tal como a API de dados, sendo as *views* utilizadas elaboradas em *Pug* e o design das páginas apresentadas foi desenvolvido com recurso à biblioteca *Bootstrap*.

4.2 Comunicação com API de Dados

Toda a comunicação deste servidor com a API de dados é efetuada com recurso à ferramenta *Axios*. Tal como foi referido anteriormente, a autenticação nesta aplicação está a ser efetuada com recurso a *tokens JWT*, sendo que estes, por sua vez, estão a ser guardados pelo *frontend* num *cookie* do *browser*. O *frontend* não faz a verificação do token, simplemente, quando existe, descodifica-o para aceder à informação neste contida. Isto é um passo de segurança: se o *frontend* não tiver acesso à palavra-passe para a verificação do *token*, existe uma menor superfície de ataque na aplicação.

4.3 Armazenamento de Ficheiros

Tal como foi referido no ínicio do relatório o armazenamento dos ficheiros associados a cada recurso está a ser efetuado no servidor *frontend*. De modo a nunca atingirmos o limite máximo de ficheiros por pasta, seguindo a sugestão do professor, decidimos criar uma série de sub pastas onde depois guardamos os ficheiros. Estas sub pastas são criadas com base na data em que o recurso é introduzido na aplicação e no id que é atribuído ao próprio recurso, criando um caminho com o seguinte formato: fileStore/ano/mês/dia/id do recurso/.

4.4 Upload e Download de Bags

Visto que é neste servidor que são armazenados os ficheiros, é este o responsável pelo download e upload dos *Bags*. No processo de download começamos por criar o *Bag* correspondente ao recurso, neste são incluídos todos os ficheiros associados ao recurso em questão. Além destes ficheiros é também inserido um ficheiro chamado *info.json* onde guardamos toda a informação relevante para podermos depois fazer o upload deste. Neste ficheiro não são guardados os dados referente ao utilizador que carregou o ficheiro, nem os comentários do recurso, pois no momento de upload, o ID do utilizador que carrega e os IDs dos utilizadores que comentaram este recurso poderiam não ser válidos. Por fim, o *Bag* é comprimido e é efetuado o download do mesmo.

O processo de upload de um *Bag* na plataforma começa pela descompactação do ficheiro. De seguida, lemos o ficheiro *info.json*, e vamos verificar que todos os ficheiros nele mencionados existem e não foram modificados após a criação do *Bag* e fazemos o armazenamento destes no local adequado. O utilizador que efetuar o upload do recurso passa a ser o seu dono e os commentários aparecem vazios.

4.5 Instruções de execução

1. Assegurar que temos a versão mais recente do *NodeJS* instalada (o servidor foi testado no *NodeJS* versão 15.3.0);
2. Efetuar *npm install* para instalar todos os pacotes necessários;
3. Executar o servidor:
    - Em modo de desenvolvimento - *npm run start:dev*;
    - Em modo de produção - *npm start*.

Este servidor não requer povoamento inicial, bastando portanto o povoamento descrito no capítulo anterior.

Capítulo 5

Povoamento da Aplicação

Neste capítulo irá ser descrito o processo de povoamento da base de dados. Existem dois tipos de povoamento: um deles é obrigatório, e o outro não o é.

5.1 Povoamento obrigatório

O povoamento obrigatório inicializa a base de dados do *MongoDB* com certos documentos que são necessários ao bom funcionamento da base de dados. Especificamente, são criados os documentos de *roles* necessários ao funcionamento da aplicação e do sistema de permissões.

Para efetuar o povoamento basta, com o *MongoDB* ligado, navegar até ao diretório *api-server/datasets/* e recorrer ao seguinte comando:

```
$ mongoimport -db DAW2020Projeto -file userroles.json -jsonArray
```

O servidor está agora pronto para aceitar o registo de utilizadores.

5.1.1 Criação de um administrador

Existem duas formas de criar um administrador:

1. Criar um utilizador, normalmente, através da interface gráfica, e posteriomente editar manualmente o ID da *role* que lhe está atribuída, diretamente na base de dados;

2. No modo de desenvolvimento, é possível chamar a rota */auth/registerteste* na API de dados. Esta rota cria automaticamente um utilizador com as credenciais *d12@gmail.com* e *teste*. Escusado será dizer que este não é o modo preferido para um ambiente de produção, e que foi apenas criado para facilitar os testes da aplicação.

5.2 Povoamento opcional

Para efeitos de teste, pode ser necessária a criação de vários recursos na base de dados. Este povoamento da aplicação com vários recursos pode ser feito com recurso ao *script Python* incluído em *api-server/datasets/gits.py*. Este *script* está programado para fazer *download* automático da maior parte dos repositórios *GitHub* dos alunos de PRI2020 e de DAW2020, tanto dos testes como dos TPCs efetuados durante as aulas do semestre. No entanto, os ficheiros obtidos têm uma ordem de grandeza grande, pelo que a sua inclusão na entrega final do projeto está fora de questão.

O *script* em questão utiliza a API do *GitHub* para efetuar o *download* dos repositórios em questão como um ficheiro ZIP. Estes ficheiros são depois tratados e convertidos para um *Bag* que pode ser aceite pela página de *upload* da nossa aplicação.

Para correr o script, é necessário ter o *Python 3.8* instalado. É também necessário ter uma chave da API do *GitHub* gerada, caso contrário não vai ser possível obter todos os ficheiros de uma só vez. A chave pode ser colocada na respetiva variável, como uma string, dentro do ficheiro, bem como o nome do utilizador do *GitHub* a quem corresponde a chave.

Assim, basta instalar as dependências:

```
$ pip install -r requirements.txt
```

E podemos agora proceder à execução do *script*:

```
$ python gits.txt
```

O *script* irá gerar uma pasta chamada *files* que contém todos os *Bags* que acabaram de ser transferidos. Estes *bags* podem agora ser carregados através da interface gráfica da aplicação. Apesar da interface permitir o carregamento de todos os ficheiros de uma só vez, recomendamos que apenas sejam selecionados, no máximo, entre 40 a 50 *bags*, para prevenir que o servidor fique sem RAM, e também para permitir que o *browser* não dê *timeout* no pedido.

Capítulo 6

Conclusão

Durante a realização deste trabalho tivemos a oportunidade de aprofundar os nossos conhecimentos relativos ao funcionamento e ao desenvolvimento de aplicações Web modernas, nomeadamente às tecnologias *Node* e *Express*. Aprendemos também a utilizar as mesmas para construir APIs de dados de acordo com as melhores práticas da indústria, e também aprendemos a desenvolver aplicações de *frontend* com recurso a *frameworks* modernas e responsivas.

Tal como está, a aplicação cumpre todos os requisitos pedidos pelo professor no enunciado deste projeto. No entanto, sabemos que existe ainda muito trabalho que poderá ser feito neste projeto, como por exemplo:

- Melhorar significativamente a interface Web, para ser mais intuitiva;
- Melhorar o tratamento de erros do *backend*;
- Melhorar o modo como os ficheiros dos recursos são tratados e guardados, i.e., retirar o seu processamento do servidor de *frontend* e mover para a API de dados ou para um outro servidor dedicado exclusivamente a isso. Deste modo é possível separar a API de dados da interface Web, fazendo com que seja possível aceder aos dados do sistema programaticamente e sem ser necessário passar pela interface Web;
- Melhorar significativamente a legibilidade do código produzido, bem como a separação de responsabilidades do mesmo;

No futuro poderão ser atacados alguns destes pontos, com especial destaque para o primeiro, de melhoramento da interface Web.

Bibliografia

[1] *Model One-to-Many Relationships with Document References*. https://docs.mongodb.com/manual/tutorial/model-referenced-one-to-many-relationships-between-documents/. Acedido a 7 de fevereiro de 2021.