



UNIVERSIDADE DO MINHO

MESTRADO EM ENGENHARIA INFORMÁTICA

Trabalho Prático

Fábio Gonçalves, PG42827

Joel Costa Carvalho, PG42837

Vasco António Lopes Ramos, PG42852

Desenvolvimento Aplicações WEB

4º Ano, 1º Semestre

Departamento de Informática

7 de fevereiro de 2021

Índice

1	Introdução	1
1.1	Contextualização	1
1.2	Estrutura do relatório	1
2	Requisitos e Arquitetura	2
2.1	Requisitos	2
2.2	Arquitetura	3
2.2.1	Arquitetura Conceptual	3
2.2.2	Arquitetura de Instalação	3
3	Decisões de Implementação	5
3.1	Modelo de Dados	5
3.1.1	Utilizadores	5
3.1.2	Recursos	6
3.2	Estratégias de Autenticação e Autorização	7
3.3	Validação da Submissão (<i>import</i>) de Recursos	8
3.4	Estratégia de Armazenamento dos Recursos	9
3.5	Mecanismo de Notificações	10
3.6	Recuperação da <i>Password</i>	11
4	Como executar/utilizar a plataforma	13
4.1	Utilização local	13
4.2	Utilização <i>online</i> (<i>cloud</i>)	13
4.3	<i>Dataset</i>	14
4.3.1	Fase 1: Criação de Utilizadores	14
4.3.2	Fase 2: Criação de Recursos	15
5	Resultado Final	17
5.1	Registo	17
5.1.1	Registo <i>Google</i>	18
5.2	<i>Feed</i> de Recursos	20
5.3	<i>Rating</i>	21

5.4	Filtragem e <i>Hashtags</i>	22
5.5	Comentários	24
5.6	<i>Backoffice</i>	25
6	Conclusão	28
A	Definição do modelo de dados associado aos utilizadores	29
B	Definição do modelo de dados associado aos recursos	30
C	Função de autenticação Local	32
D	Função de autenticação <i>JWT</i>	33
E	Função de autenticação <i>Google</i>	34
F	Função de validação '<i>BagIt</i>'	35
G	Excerto do mecanismo de armazenamento dos recursos	36
H	<i>Pipeline</i> de CI/CD no <i>GitHub</i>	37
	Referências	38

Lista de Figuras

1	Arquitetura Conceptual (3 camadas)	3
2	Arquitetura de Instalação	4
3	Recuperação <i>Password</i>	11
4	Localização da pasta <i>uploads</i>	16
5	Registo com erros	18
6	<i>Página inicial com autenticação</i>	19
7	<i>Contas Google</i>	19
8	<i>Registo pelo Google</i>	20
9	<i>Feed de Recursos</i>	21
10	<i>Rating dos Recursos</i>	21
11	<i>Rating dos Recursos</i>	22
12	Filtragem <i>Desktop</i>	22
13	Filtragem <i>Mobile</i>	23
14	<i>Hashtags</i>	24
15	<i>Comentários</i>	25
16	<i>Dashboard</i>	26
17	<i>Listagem de Utilizadores</i>	27
18	<i>Listagem de Recursos</i>	27

Lista de Códigos

1	Exemplo da estrutura e conteúdo do ficheiro <i>manifest.json</i>	9
2	Comandos para dar <i>import</i> dos <i>datasets</i> no <i>MongoDB</i>	15
3	Definição do modelo de dados associado aos utilizadores	29
4	Definição do modelo de dados associado aos recursos	30
5	Função de autenticação Local	32
6	Função de autenticação <i>JWT</i>	33
7	Função de autenticação <i>Google</i>	34
8	Função de validação <i>'BagIt'</i>	35
9	Excerto do mecanismo de armazenamento dos recursos	36
10	<i>Pipeline</i> de CI/CD no <i>GitHub</i>	37

1 Introdução

1.1 Contextualização

No âmbito da UC de Desenvolvimento Aplicações WEB foi-nos proposta a realização de um projeto final que consiste no desenvolvimento de um repositório de recursos/ficheiros, do tipo *moodle* com funcionalidades de redes sociais, tais como, avaliar e comentar os *posts* existentes na plataforma, bem como a integração com um sistema de notificações/avisos de atividade na mesma. Um requisito também bastante importante era que a plataforma seguisse uma conceptualização arquitetural que respeitasse o *standard Open Archival Information System (OAIS)*.

1.2 Estrutura do relatório

Este relatório encontra-se dividido em 6 capítulos.

No capítulo 1 é feita uma breve introdução a este trabalho, e é explicada a estrutura deste relatório.

No capítulo 2 é feita uma breve revisão dos requisitos associados a este trabalho prático e uma breve apresentação das arquiteturas conceptual e de instalação associadas à nossa solução.

No capítulo 3 são explicadas as decisões de implementação associadas aos pontos mais importantes e relevantes do nosso trabalho.

No capítulo 4 é feita uma breve exposição sobre quais são os mecanismos para testar e executar a nossa aplicação.

No capítulo 5 é feita uma apresentação do resultado final do nosso trabalho, com especial foco na utilização da plataforma, através da camada de apresentação (*interface*).

No capítulo 6 é feita uma breve conclusão do trabalho realizado, onde são retiradas conclusões sobre o mesmo e são apontadas sugestões para possível trabalho futuro de forma a melhorar a solução em análise.

2 Requisitos e Arquitetura

2.1 Requisitos

Tendo em conta os requisitos expostos pelo docente durante a apresentação do projeto, os requisitos principais nos quais o processo de desenvolvimento se focou foram os seguintes:

- O cumprimento com o *standard OAIS*, no qual existem 3 tipos de atores (produtor, consumidor e administrador).
- Disponibilização de recursos educativos de vários tipos.
- Possibilidade de um utilizador adicionar novos recursos.
- Classificação de recursos por ano, tipo e categorias (*hashtags*).
- Possibilidade de um utilizador fazer um *Post* sobre um recurso.
- Possibilidade de os utilizadores comentarem *Posts*.
- Possibilidade de os utilizares avaliarem *Posts*, através de um sistema de *Ranking*.
- Validar todas as submissões seguindo um mecanismo semelhante ao *BagIt*, documentação disponível em [1].
- Permitir a exportação de recursos no formato semelhante ao da submissão (formato *BagIt*).
- O sistema deverá estar protegido com autenticação (*username+password*, chave *API*, *Google*).
- Os utilizadores deverão ter a possibilidade de interagir com um sistema de notificações/avisos relativos a nova atividade na plataforma.

2.2 Arquitetura

2.2.1 Arquitetura Conceptual

A arquitetura conceptual da solução desenvolvida, isto é, a arquitetura relacionada ao desenvolvimento do código foi desenhada no sentido de separar o sistema em 3 camadas (apresentação, lógica e dados).

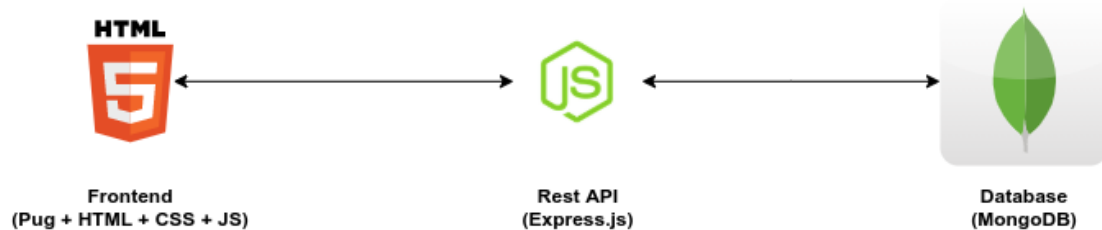


Figura 1: Arquitetura Conceptual (3 camadas)

Como se pode ver na figura 1, a camada mais externa é a de apresentação (**Frontend**), que foi desenvolvida com *Pug*, *HTML*, *CSS* e *JavaScript*. De seguida temos a camada onde toda a lógica do sistema está contida, incluindo lógica de autenticação, (**Rest API**), que foi desenvolvida com a *framework Express.js*, baseada em *Node.js*. E, por fim, a camada de dados que foi desenhada em *MongoDB*.

2.2.2 Arquitetura de Instalação

Ao nível da arquitetura de instalação a situação já não é a mesma. De forma a manter a simplicidade, até para, no fim, ser mais realista o nosso objetivo de querer deixar a plataforma disponível *online*, num serviço de *cloud*, decidiu-se ter um único servidor com as camadas de *Frontend* e *Backend*. A figura 2 permite ver a arquitetura de instalação descrita.



Figura 2: Arquitetura de Instalação

A decisão de ter as duas camadas no mesmo servidor aplicacional em nada afetou a arquitetura descrita na secção anterior, pois, apesar de todo o código ter sido desenvolvido num único servidor, manteve-se sempre a separação conceptual entre o que é camada de apresentação e camada de lógica de negócio, pelo que estas duas camadas são bastante independentes uma da outra e o esforço de separá-las em servidores (aplicações) diferentes, num trabalho futuro, será muito reduzido. Isto foi possível de se fazer, pois, na camada de apresentação todos os dados eram sempre consumidos através da API de dados que era acedida pelo *package axios*, tal como se esta (a API) estivesse numa aplicação *Node.js* diferente.

3 Decisões de Implementação

3.1 Modelo de Dados

De forma a conseguirmos encontrar um modelo de dados compreensivo e capaz de preencher as necessidades da plataforma, adotámos um processo iterativo que resultou em duas coleções ("tabelas"), uma para os utilizadores e outra para os recursos. Alguns exemplos das fases intermédias, evolutivas, da construção do modelo de dados podem ser encontrados na pasta *_data/examples* do repositório.

3.1.1 Utilizadores

Para os utilizadores, e tal como se pode ver no anexo [A](#), os campos que decidimos guardar são os seguintes:

- *username*: tem de ser valor único
- *first_name*
- *last_name*
- *password*: que é guardada através de um processo de geração da respetiva *hash + salt*
- *email*: tem de ser valor único
- *is_admin*: valor booleano que representa se o utilizador é administrador ou não (utilizado na plataforma para asserção de autorização)
- *is_active*: valor booleano que representa se o utilizador tem permissão para aceder à plataforma. Por omissão é verdadeiro; passa a falso se o utilizador decidir eliminar a sua conta ou se for bloqueado por um administrador.
- *filiation.institution*: instituição a que pertence
- *filiation.position*: posição que detém dentro da instituição a que pertence
- *token*: utilizado para autenticação (null por defeito; toma valor quando o utilizador se autentica na plataforma; volta a null quando é feito um *logout*)

- *accessToken*: utilizado para a integração com a autenticação via *Google*
- *notifications*: estrutura (*array* de objetos) onde são registradas as notificações do utilizador

3.1.2 Recursos

A abordagem adotada pelo grupo foi que, cada recurso é inerentemente, um *post* (uma abordagem que combina o conceito de *post* em redes sociais como o *Facebook* e os *papers* que existem em plataformas como *Research Gate*). Deste modo, os recursos para além das informações necessárias a este, incluem informação como título, descrição, tipo, *tags*, entre outras. Assim, os campos que decidimos guardar para cada recurso, tal como se pode ver em maior detalhe no anexo B, são os seguintes:

- *path*: caminho relativo que aponta para onde está armazenado o recurso dentro do nosso sistema
- *name*: o nome do recurso quando este foi submetido (isto é, o nome do *zip*)
- *mime_type*: o tipo de ficheiro associado ao recurso, ex: *application/octet-stream* (por omissão), *text/plain*, *application/pdf*, entre outros. Este campo é de grande importância para sabermos como representar os recursos visualmente na plataforma.
- *image*: caminho relativo que aponta para onde está armazenada a imagem associada ao recurso dentro do nosso sistema (caso o utilizador não queira associar nenhuma imagem, é associada uma imagem padrão por omissão)
- *type*: o tipo de recurso que está a ser submetido, isto é, se é um livro, aplicação, relatório, etc
- *description*: texto de descrição associado ao recurso (uma espécie de 'abstracto' para o recurso)
- *author*: a pessoa que submeteu o recurso
- *year*: o ano em que o recurso foi produzido

- *size*: o tamanho, em *bytes*, do recurso
- *date_added*: a data em que o recurso foi submetido
- *last_updated*: data da edição mais recente do recurso
- *subject*: o assunto/título do recurso
- *tags*: uma estrutura (*array* de *strings*) com as *tags*/categorias associadas ao recurso
- *rating*: estrutura em que se guarda a informação associada às avaliações dos recursos (que variam entre 0 e 5)
- *comments*: estrutura (*array* de objetos de 2 níveis) onde se guardam os comentários e respostas a comentários associados aos recursos

3.2 Estratégias de Autenticação e Autorização

Como forma de um utilizador realizar a autenticação na nossa plataforma, este terá 2 formas de o fazer. A primeira passa por realizar o *login* de forma "local", onde para tal, utilizámos o *package npm passport* com duas estratégias, a *LocalStrategy* e a *JWTStrategy*, como podemos ver no anexo C e anexo D. Inicialmente o utilizador insere os dados para realizar o login e de seguida é criado um *jwt token* para ser gerada uma *cookie*. Esta garante que este *token* é válido, durante a permanência do utilizador na nossa aplicação, e que possui os requisitos necessários, como a data de expiração não vencida e o utilizador associado é o que está autenticado. Por fim e como podemos ver no anexo E, decidimos introduzir o método de autenticação pela plataforma *Google*. Para tal, utilizamos a estratégia do *passport GoogleStrategy*, tendo o utilizador a possibilidade de entrar com a sua conta na nossa aplicação. O utilizador autentica-se com a sua conta *Google* e de seguida é criado um *jwt token* para posteriormente ser gerada uma *cookie*.

Quando o utilizador decidir sair da aplicação esta *cookie* é apagada e o utilizador terá que realizar de novo a autenticação. No caso da estratégia da autenticação através do *Google* é gerado um *accessToken* que é guardado na base de dados, e quando o utilizador quiser sair da aplicação é realizado um pedido para revogar o

accessToken e, conseqüentemente terá que realizar de novo a autenticação, quando quiser voltar a entrar.

Ao nível de autorização, implementámos 2 principais tipos de validação de autorização. O primeiro nomeado por *isAdmin*, é bastante óbvio, ou seja, aqueles utilizadores que têm o papel de administrador, podem adicionar utilizadores, adicionar/remover administradores e ter acesso à *dashboard*. O segundo nomeado de *checkAuthorization*, onde os utilizadores que têm o papel de produtor e de administrador podem realizar a edição e a eliminação dos recursos criados, no caso dos produtores apenas os seus recursos, no caso do administrador todos os recursos.

3.3 Validação da Submissão (*import*) de Recursos

Por critério do docente, foi pedido que todos os recursos fossem validados quanto à sua estrutura e conteúdo. Para isso, e por sugestão do docente, o grupo baseou-se no *BagIt*, como já referido antes, para o seu mecanismo de validação.

O primeiro requisito é que todos os recursos sejam submetidos em formato **ZIP**, caso não seja enviado um *zip*, o recurso não é aceite e é devolvida uma mensagem de erro de acordo com a razão (ou razões) pela(s) qual(is) não foi aceite.

O segundo requisito é que na raiz do *zip*, venham apenas dois itens: um ficheiro chamado **manifest.json** e uma pasta chamada **data** onde devem ser incluídos todos os ficheiros e pastas que se quer submeter. O ficheiro referido (*manifest.json*) é, como o nome diz, o manifesto do recurso, onde, respeitando um formato específico, devem vir listados todos os ficheiros submetidos (exceto o próprio). Este ficheiro é usado para validar se o conteúdo enviado bate certo com o que está especificado no manifesto. Caso esta estrutura não seja respeitada ou o manifesto não tenha as informações corretas e de acordo com o que está efetivamente a ser enviado, a submissão não é aceite.

Relativamente ao ficheiro de manifesto, escolheu-se JSON para o seu tipo de ficheiro, por ser um formato extremamente simples de ler, tratar e analisar em *Javascript*. Relativamente ao seu conteúdo, a listagem de código 1 demonstra qual

deve ser a sua estrutura e que informação deve conter.

```
{
  "version": "1.0.0",
  "encoding": "UTF-8",
  "files": [
    "file1",
    "folder1/file1",
    "folder1/file2",
    "folder1/file3",
    "folder2/file1",
    (...)
  ]
}
```

Código 1: Exemplo da estrutura e conteúdo do ficheiro *manifest.json*

Como se pode ver na listagem de código 1, o manifesto deve conter três informações: a versão associada ao manifesto, o seu *encoding*, e a informação relativa a que ficheiros que estão a ser submetidos.

Os primeiros dois campos foram criados para respeitar a estrutura base do *BagIt*, contudo não são avaliados, nem validados pelo nosso sistema (de ressaltar que no exemplo apresentado existe uma gralha na palavra "*encoding*" e que se replicou em todos os recursos gerados no nossos *dataset* que passou despercebida até ao momento da escrita deste relatório, precisamente por esta valor não ser analisado ou validado).

Relativamente ao último campo, este é analisado e é validada a sua veracidade contra os ficheiros que estão a ser efetivamente submetidos no *zip*. O pedaço de código presente no anexo F permite analisar a função desenvolvida para realizar esta validação.

3.4 Estratégia de Armazenamento dos Recursos

Outro processo de relevo é o armazenamento dos recursos aquando da sua submissão, principalmente ao nível do sistema de ficheiros (devido a uma série de

preocupações relativas a limitação de ficheiros e pastas por diretório, entre outras). Assim, foi necessário encontrar uma estratégia para tentar resolver ou, pelo menos, minimizar esses problemas. É precisamente isso que vai ser discutido nesta secção.

Para tentar minimizar este problema a estratégia adotada foi a seguinte:

1. Dividir os recursos por tipo de recurso.
2. Para cada tipo de recurso, dividir os recursos por utilizador.
3. No sistema de ficheiros, por tipo de recurso e utilizador, cada recurso tem um diretório próprio associado, sendo que o nome desse diretório é o *timestamp* do momento em que o recurso é submetido.

O pedaço de código presente no anexo [G](#) apresenta um excerto do processo de armazenamento que representa o mecanismo acima descrito.

3.5 Mecanismo de Notificações

Decidimos que seria muito importante, numa plataforma desta categoria, termos um mecanismo de notificações. Para tal recorreremos ao *package* ***socket.io***. Este pacote, na sua definição, permite enviar mensagens para todos os utilizadores autenticados. Adaptámos esta estratégia de comunicação de forma a que, sempre que um utilizador insira um recurso, todos os utilizadores autenticados, exceto o próprio, irão receber, não só uma notificação pelo *browser*, para tal terá que permitir o envio de notificações por este meio, mas também o envio de uma mensagem para a secção das notificações presentes no menu. Também implementámos estas notificações para dois eventos importantes na plataforma, que são: a inserção de um novo comentário e a atribuição de uma classificação num recurso. Neste primeiro, quando alguém insere um novo comentário numa publicação, o autor desta, irá ser alertado com uma mensagem que indica que alguém escreveu um comentário e o seu nome. Quanto à classificação do recurso, o autor do recurso irá receber do mesmo modo uma mensagem que indica o número da classificação e o respetivo utilizador que classificou o recurso.

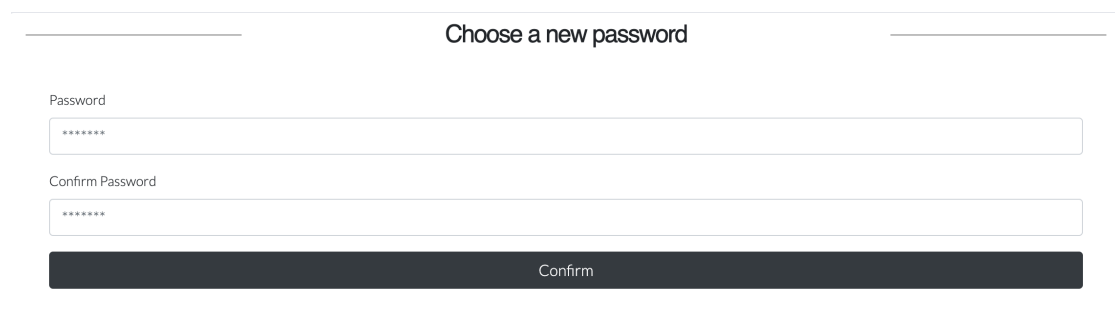
Aquando da chegada de uma notificação o utilizador é alertado, não só pelo *browser* mas também na secção das notificações (um ícone de um sino no menu),

visto que colocámos uma animação nesta secção que se destaca aquando destes comportamentos (o sino vibra constantemente e muda de cor).

3.6 Recuperação da *Password*

De forma a qualquer utilizador recuperar o acesso total à sua conta, foi implementado a *feature* de Recuperação de *Password*. Tendo em consideração as aplicações comuns, optámos pela requisição do email registado na plataforma e consequentemente o envio de um email com os detalhes para a recuperação da mesma.

Após a verificação do email na plataforma é gerado um *token* e uma *cookie*, com tempo limitado, de forma a garantir um processo minimamente seguro. No email enviado segue um *link*, do tipo `.../auth/recoverPassword/:token`, ver figura 3, que, ao ser acedido, verifica a autenticidade do *token* com a *cookie* e consequentemente permite alterar a *password*.



The image shows a web form titled "Choose a new password" centered at the top. Below the title, there are two input fields. The first is labeled "Password" and contains six asterisks. The second is labeled "Confirm Password" and also contains six asterisks. At the bottom of the form is a dark grey button with the word "Confirm" in white text.

Figura 3: Recuperação *Password*

Para a utilização deste fluxo usufruímos do *package* *express-mailer*. A Tabela 1, representa as configurações base dos *providers* mais comuns. Nota para o funcionamento correto de alguns, configurações adicionais são requisitadas, como o *Gmail* que necessita de dar acesso a *apps* menos seguras.

Configurações SMTP

Provider	Configurações
Gmail	smtp.gmail.com - 465
Office365	smtp.office365.com - 587
Outlook	smtp-mail.outlook.com - 587

Tabela 1: Configurações SMTP

4 Como executar/utilizar a plataforma

Desde o início do trabalho que o objetivo foi sempre deixar o processo de teste e utilização da plataforma o mais simples e rápido possível. Desse modo, este capítulo tem por objetivo discriminar quais as abordagens para simplificar os processos associados a instalar, executar e testar a plataforma e também instruções sobre como executar e testar a mesma.

4.1 Utilização local

Para instalar localmente a aplicação, é primeiro necessário ter o *MongoDB* instalado (documentação oficial para instalação disponível em: [2]).

Após instalar o *MongoDB*, é necessário descarregar a aplicação, que se encontra no *GitHub* e instalar todas as dependências da aplicação (os *node packages*). Tal pode ser feito com o comando `npm install`.

O último passo é, efetivamente, executar a aplicação. Esta execução pode ser feito em modo de **desenvolvimento** (com mecanismos de *hot refresh*, através do *package nodemon*), com o comando `npm run develop`, ou em modo de **produção**, com o comando `npm start`.

Estas instruções apenas garantem a execução da plataforma, para informações sobre como pré-popular o sistema com um conjunto grande de dados ver a secção 4.3.

4.2 Utilização *online* (*cloud*)

Para além da execução local, e de forma a permitir uma utilização e teste da plataforma mais simples e acessível, decidiu-se disponibilizar a plataforma *online*. Para tal utilizou-se dois serviços de *Platform* e *Software as a Service*.

Para a base de dados, *MongoDB*, usou-se um serviço da mesma organização responsável pelo *MongoDB*, o *MongoDB Atlas*, na sua camada de utilização livre de custos que nos permite ter a nossa base de dados *Mongo* online e disponível para consumo.

Para a aplicação, utilizou-se a camada livre de custos do *Heroku*, através de uma *pipeline* de *CI/CD* do *GitHub* que nos permite dar *deploy* da nossa aplicação sempre que existe um *commit* na *branch* ***production***, tal como se pode ver no anexo H. A plataforma encontra-se, deste modo, disponível no seguinte [link](#).

É, no entanto, de ressaltar que pelo facto da camada *free* do *Heroku* disponibilizar apenas um sistema de ficheiros efémero que dura, no máximo, um dia, a plataforma que disponibilizamos no [link](#) acima não foi populada com o *dataset* gerado por nós, já que passado umas horas de cada carregamento toda e qualquer informação que estivesse guardada no sistema de ficheiros associado à nossa aplicação teria desaparecido e deixaria de estar disponível.

4.3 *Dataset*

A última secção deste capítulo relaciona-se com a construção do *dataset* para testar a plataforma desenvolvida. Para o desenvolvimento deste processo de construção de um *dataset* definimos dois objetivos:

- Ter cerca de 5.000 utilizadores.
- Ter cerca de 15.000 recursos.

Para tal, desenvolveu-se um processo com duas fases distintas: a primeira relacionada com a criação de utilizadores e a segunda relacionada com a criação dos recursos. Para ambas as fases foi utilizado o *package* ***faker*** para a criação de informação.

4.3.1 Fase 1: Criação de Utilizadores

Esta fase é totalmente automática, isto é, basta estar localizado dentro da pasta `_data` e executar o comando `npm run create-users`. Este *script* desenvolvido em *Node.js* irá iterar 5000 vezes e criar um utilizador em cada iteração. Como já mencionado acima, todos os dados (nomes, email, password, entre outros) são gerados através do *package* ***faker***. No fim, a lista de utilizadores criados, com todas as suas informações, é escrita, em formato JSON, para um ficheiro.

4.3.2 Fase 2: Criação de Recursos

A segunda fase, relacionada com a criação de recursos não é totalmente automatizada como a secção descrita cima. Este tem um primeiro processo: a criação dos *bags*, isto é, dos *zips* com os recursos e a meta-informação necessária para cumprir com o protocolo de *upload* adotado pelo grupo. Para este processo foram também criados *scripts* para auxiliar o processo da criação dos manifestos e da compactação em *zip*, contudo, a criação da estrutura correta dentro da pasta do recurso continua a ser manual.

Tendo os *zips* preparados, o processo de inserção é totalmente automático, com a execução do comando `npm run create-resources`. Para a submissão dos recursos, a nossa abordagem foi criar uma *pool* de *zips* válidos com cerca de 92 *zips* com diversos tamanhos, que variam entre os 700 *bytes* e os 100 *megabytes*. Ao longo do *script*, reutilizando os *zips* criados, em cada submissão é escolhido aleatoriamente um *zip* dos 92 existentes na nossa *pool* de recursos e, então, submetido na plataforma.

Após a conclusão da população, os *datasets* dos utilizadores e recursos no *MongoDB* foram exportados para ficheiros JSON e a pasta da nossa plataforma onde todos os recursos são armazenados foi compactada num *zip*. Todos estes dados estão disponíveis no seguinte [link](#), não tendo sido colocados no nosso repositório *git* devido ao seu tamanho.

Para utilizar o dataset criado pelo grupo, basta dar *import* dos ficheiros JSON para uma base de dados no *MongoDB* com o nome de `daw_project`, sendo que a coleção dos utilizadores se deve chamar `users` e a dos recursos `resources`. A porção de código 2 exemplifica como proceder ao *import* referido.

```
mongoimport --db daw_project \
  --collection users --file users.json --jsonArray

mongoimport --db daw_project \
  --collection resources --file resources.json --jsonArray
```

Código 2: Comandos para dar *import* dos *datasets* no *MongoDB*

Por fim, basta apenas descompactar o *zip* chamado **uploads.zip** e colocar a pasta com o nome *uploads* que existe dentro desse *zip*, dentro da pasta *app* da aplicação desenvolvida (isto é, ao mesmo nível, por exemplo, da pasta *public* e do ficheiro *app.js*). A figura 4 demonstra onde deverá ficar a referida pasta ***uploads***.

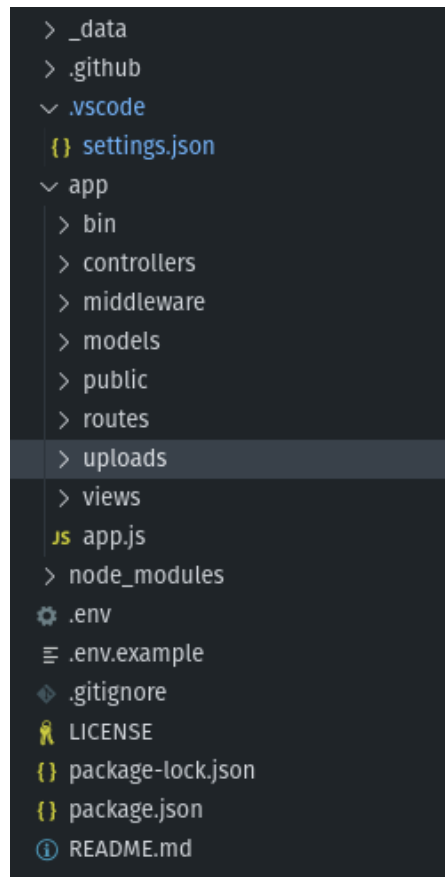


Figura 4: Localização da pasta ***uploads***

5 Resultado Final

5.1 Registo

Uma das fases mais importantes do desenvolvimento foi o registo, esta permite aos futuros utilizadores se registarem na plataforma de forma assíncrona onde é possível despoletar erros baseado na ações não desejadas pelo utilizador.

Posto isto, o registo é composto pelos seguintes campos mandatórios:

Registo Manual		
Campo	Tipo	Requisitos
Primeiro Nome	Texto	-
Último Nome	Texto	-
<i>Username</i>	Texto	Min 2 Caracteres - Único
Email	Email	Único
Instituição	Texto	Min 2 Caracteres
Posição	Texto	Min 2 Caracteres
<i>Password/Confirmar Password</i>	<i>Password Password</i>	Min 2 Caracteres - Min 1 Maiúscula Min 1 Minúscula

Tabela 2: Registo Manual

Caso o utilizador não cumpra os requisitos, mensagens informativas serão demonstradas, como comprova a figura 5. Em caso de sucesso, o *login* é realizado no imediato.

Register

First Name

Joe

First Name field is required.

Last Name

Brown

Last Name field is required.

Username

jb_12345

Username field must be at least 2 chars long.

Email

pg12345@alunos.uminho.pt

Email field must be an email.

Institution

Univ. Minho

Institution field must be at least 2 chars long.

Position

student

Position field must be at least 2 chars long.

Password

Password must be complex! At least 8 characters with lowercase, uppercase and digits.

Confirm Password

Confirm Password field is required.

Register

Figura 5: Registo com erros

Ainda relativamente à *password*, definimos uma *blacklist* de *passwords* do tipo: *Passw0rd* ou *Password123*, obrigando o utilizador a compor uma chave de acesso segura. Para a realização desta etapa utilizamos dois *packages*, o *express-validator* e o *password-validator*.

5.1.1 Registo *Google*

O registo através do *Google* segue a mesma filosofia do registo referido no tópico anterior. Inicialmente o utilizador irá se deparar com um formulário para realizar a autenticação, caso não tenha conta registada poderá então criar. Caso se queira autenticar através do *Google* este terá que pressionar o botão associado, *Sign in with Google*.

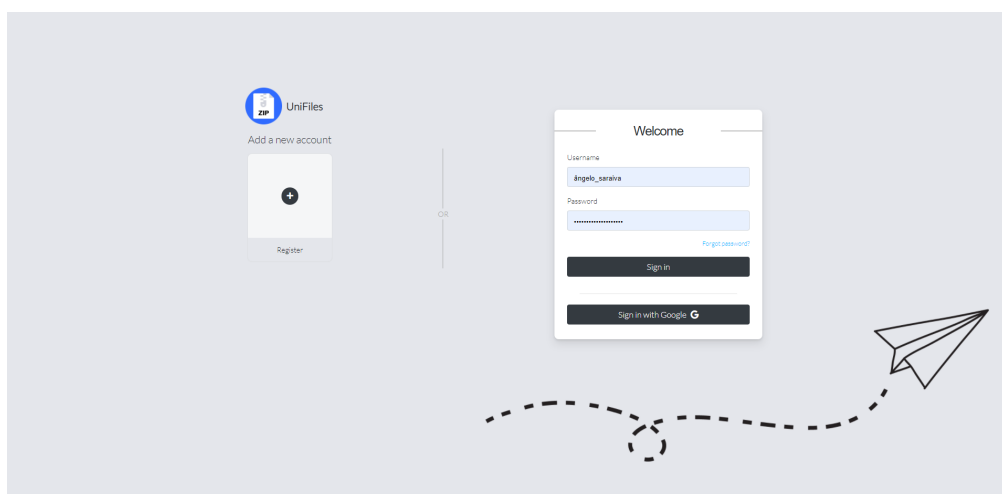


Figura 6: *Página inicial com autenticação*

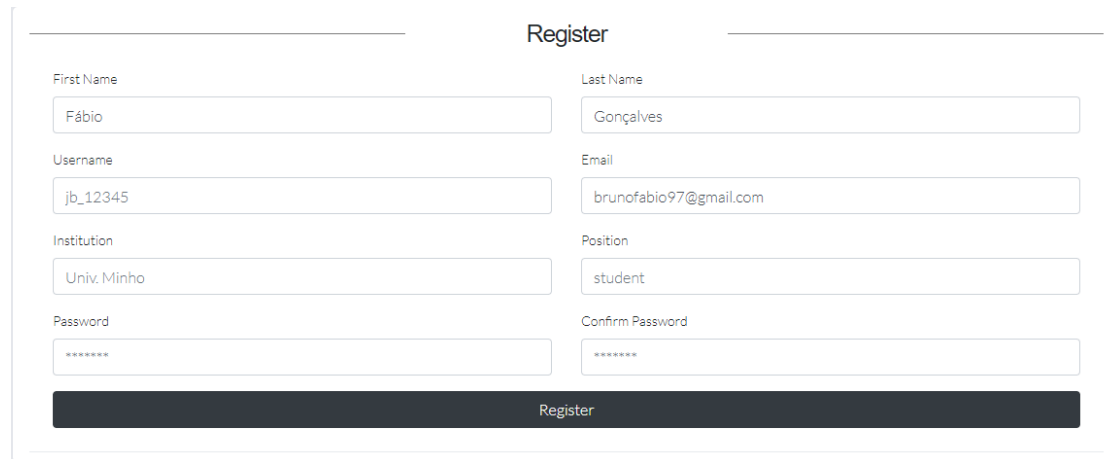
De seguida irá introduzir uma conta pessoal para poder entrar na plataforma.



Figura 7: *Contas Google*

Após a inserção dos dados pessoais, caso a conta já esteja registada, este irá automaticamente para a página principal da plataforma. Caso contrário, o utilizador será redireccionado para a página de registo, onde irá encontrar alguns

campos já pré-definidos, de acordo com a sua conta anteriormente inserida na plataforma *Google*.



The image shows a registration form titled "Register". It contains the following fields and values:

Field	Value
First Name	Fábio
Last Name	Gonçalves
Username	jb_12345
Email	brunofabio97@gmail.com
Institution	Univ. Minho
Position	student
Password	*****
Confirm Password	*****

At the bottom of the form is a dark button labeled "Register".

Figura 8: *Registo pelo Google*

5.2 *Feed* de Recursos

Presente na página mais importante da aplicação, esta será a primeira página após o registo, onde é pretendido cativar todos os intervenientes do *website*. Baseada em *Infinite Scroll*, esta tem como objetivo listar, do mais recente para o mais antigo, todos os recursos presentes na plataforma. À medida que o *scroll* é realizado, mais 5 recursos são adicionados à listagem, onde cada recurso lista detalhes como: autor, tipo, ano, título, *tags*, entre outros. A figura 9 apresenta o alinhamento dos mesmos.

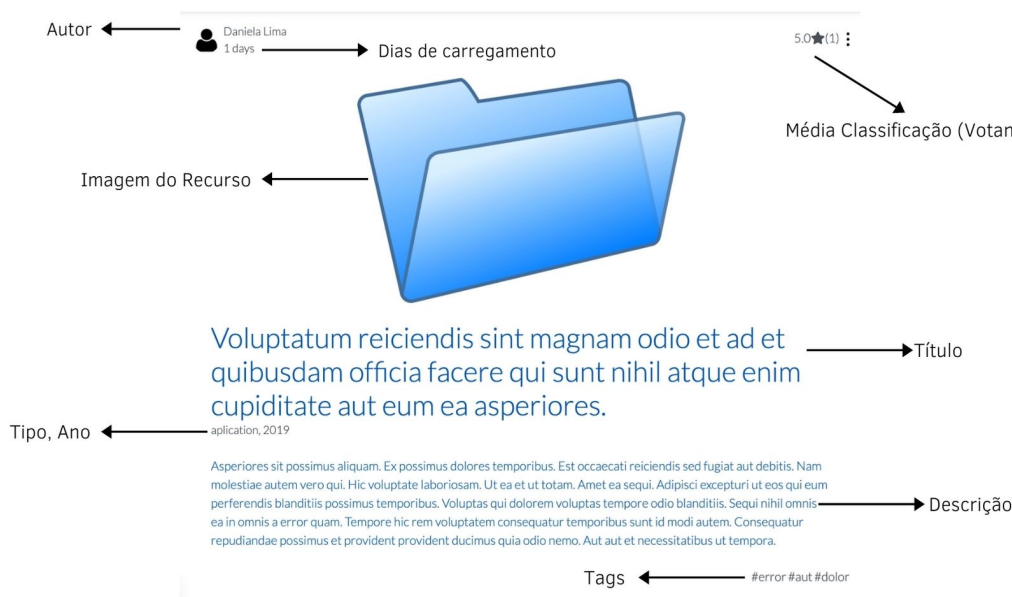


Figura 9: *Feed* de Recursos

5.3 *Rating*

Mantendo os ideais comuns da classificação, os utilizadores podem avaliar os recursos numa escala de 0 a 5. Para isso, os utilizadores deverão seleccionar qual o recurso que pretendem avaliar e seleccionar uma estrela baseada na escala definida, como observamos na figura 10.

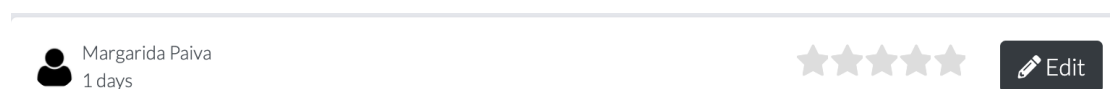


Figura 10: *Rating* dos Recursos

Após a selecção final, as estrelas são preenchidas baseadas na média dos demais votantes, representado na figura 11.



Figura 11: *Rating* dos Recursos

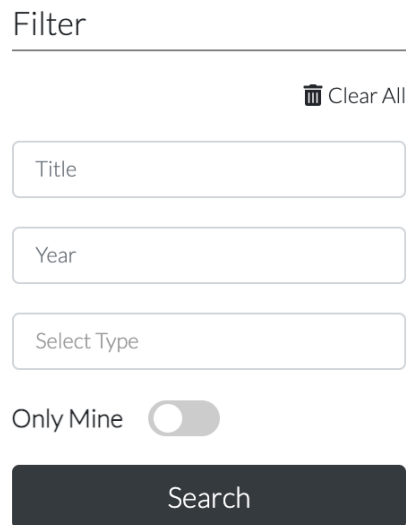
5.4 Filtragem e *Hashtags*

A listagem filtrada, ver figura 12 versão *desktop* e ver figura 13 versão *mobile*, é composta por várias combinações que, estão ao dispor dos utilizadores em 4 filtros:

- Título - Capaz de procurar qualquer recurso que contenha determinada letra(s);
- Ano - Capaz de filtrar recursos por um determinado ano;
- Tipo - Capaz de seleccionar entre 1 a 5 tipos existentes: artigos, aplicações, teses, livros e relatórios;
- Apenas 'Os Meus' - Listagem privada dos recursos inseridos pelo próprio.



Figura 12: Filtragem *Desktop*

The image shows a mobile application interface for filtering search results. At the top, the word "Filter" is displayed in a dark font, followed by a horizontal line. Below the line, there is a "Clear All" button with a trash can icon. Underneath, there are three input fields: "Title", "Year", and "Select Type". Below these fields is a toggle switch labeled "Only Mine", which is currently turned off. At the bottom of the filter section is a dark, rounded rectangular button labeled "Search".

Filter

Clear All

Title

Year

Select Type

Only Mine ☐

Search

Figura 13: Filtragem *Mobile*

Adicionalmente, há um quinto filtro, onde o utilizador pesquisa por uma *hashtag* dando a possibilidade de agregar aos filtros descritos anteriormente. Na figura 14, demonstramos uma porção de uma pesquisa baseada na *#error*.

Não havendo ocorrência de recursos baseada no *merge* dos filtros selecionados será despoletada uma mensagem, informando o utilizador que não foram encontrados recursos.



Figura 14: *Hashtags*

5.5 Comentários

De forma a obter *feedback* por parte dos intervenientes, a aplicação permite a qualquer utilizador adicionar comentários aos recursos. Após a inserção de um comentário é possível reverter a ação ou responder aos demais. Cada comentário é composto pelo autor, descrição e há quanto tempo foi postado, a figura 15 re-

apresenta um caso em que, o utilizador Joel Carvalho efetuou um comentário há 2 horas e a Margarida Paiva respondeu há 10 minutos. De notar que o utilizador logado é a Margarida e apenas consegue eliminar o seu próprio comentário através do ícone situado no canto superior direito da imagem.



Figura 15: *Comentários*

5.6 *Backoffice*

Utilizadores assinalados como administradores têm múltiplas permissões comparativamente com utilizadores normais. Os intitulados de *admins* têm possibilidade de editar e apagar qualquer *post*, além disto ainda possuem acesso a 3 páginas exclusivas: *Dashboard*, *Gestão de Utilizadores* e *Gestão de Recursos*.

- *Dashboard* - Detalha estatísticas sobre a quantidade de utilizadores ativos, *tags* mais utilizadas e um gráfico que discrimina o número de recursos postados ao longo do tempo, agrupado-os por dia ou por mês. Analisando a figura 16 e baseado no *dataset* final, obtemos um total de 2.474 utilizadores ativos, em que a *tag* mais utilizada foi *#et* e o utilizador que publicou mais conteúdo foi a Lorena Alves. Relativamente aos recursos, notamos que houve um pico de *posts* no final do ano de 2020 e o inverso acontece em Fevereiro do mesmo ano.
- *Gestão de Utilizadores* - Observando a figura 17, notamos que é possível

obter a listagem global dos utilizadores, dando a possibilidade de adicionar novos, e acções como atribuição/remoção de permissões de administrador e ativação/desativação de contas são de fácil execução.

- Gestão de Recursos - Por fim, a figura 18 apresenta a listagem global dos recursos, associados aos *links* de visualização e edição dos mesmos.



Figura 16: *Dashboard*

List of Users						
Add new						
Show	10	▼	entries	Search: <input type="text"/>		
Name	Email	Institution	Position	Is Active	Is Admin	
Adriana Oliveira	adriana_oliveira@outlook.com	Teixeira, Neves and Reis	National Program Executive		✓	
Adriana Pereira	adriana73@aeiou.pt	Matos - Sousa	Global Applications Assistant	✓		
Adriana Pires	adriana24@gmail.com	Batista Group	Global Operations Consultant			
Adriana Saraiva	adriana39@aeiou.pt	Correia - Nunes	Central Brand Developer		✓	
Adriana Simões	adriana61@yahoo.com	Nogueira - Marques	Global Usability Designer	✓	✓	
Adriana Teixeira	adriana.teixeira@gmail.com	Pereira Group	Future Usability Analyst		✓	
Adriana Torres	adriana_torres96@yahoo.com	Vieira Inc	Forward Implementation Planner			
Adriana Vieira	adriana.vieira89@outlook.com	Mendes - Moura	Legacy Research Architect			
Afonso Almeida	afonso99@outlook.com	Matias LLC	Dynamic Program Consultant	✓		
Afonso Correia	afonso_correia@portugalmail.pt	Esteves Group	Customer Web Designer		✓	
Showing 21 to 30 of 4,885 entries			Previous	1	2	3

Figura 17: *Listagem de Utilizadores*

List of Resources

Show 10 entries

Search:

Author	Type	Subject	Year	Options
Adriana Carvalho	aplication	Sit est et sapiente aut aut eligendi maiores et repellat sequi asperiores delectus sint consequuntur quam dolore et voluptatibus saepe quia dolor est distinctio illum architecto ea error.	2020	
Adriana Carvalho	book	Et enim perferendis necessitatibus labore quaerat odio.	2021	
Adriana Faria	book	Quo quia reprehenderit rerum fugiat dolore explicabo aut exercitationem minus a ut tempore dolores ut ullam deleniti excepturi exercitationem fugiat eum pariatur iure.	2018	
Adriana Faria	thesis	Nihil qui inventore dignissimos sed ullam doloribus est qui doloribus provident aut aliquid enim quos mollitia hic voluptates sed quis illum velit eos soluta id et saepe qui.	2016	
Adriana Faria	report	Necessitatibus iusto eum tempore labore eum veritatis quia aperiam molestiae neque natus minus rem veritatis qui voluptatibus itaque deserunt delectus voluptas mollitia eos vero.	2017	
Adriana Faria	book	Voluptatem quo rerum occaecati deserunt velit vel minima odio qui quia.	2019	
Adriana Faria	thesis	Voluptatem pariatur dolore voluptas nesciunt adipisci eaque iste.	2019	
Adriana Faria	thesis	Aut laborum repellat maiores voluptatem rerum molestiae architecto quod.	2020	
Adriana Gonçalves	report	Voluptatibus id reprehenderit fuga ipsam dolore nihil fugiat sit itaque dolor vero nihil.	2017	
Adriana Gonçalves	book	Et similique voluptates repellat veniam consequatur voluptatibus non.	2015	

Showing 31 to 40 of 15,022 entries

Previous

1

2

3

4

5

...

1,503

Next

Figura 18: *Listagem de Recursos*

6 Conclusão

Através da realização deste projeto, foi possível aplicar e consolidar os diversos conhecimentos que nos foram sendo passados ao longo de todo o semestre, nomeadamente nas aulas teórico-práticas da Unidade Curricular de Desenvolvimento de Aplicações *Web*. Apesar de já familiarizados com estas tecnologias, este *website* obrigou-nos a explorar novas estratégias de implementação.

As maiores adversidades encontradas foram ao nível do mecanismo das notificações, visto que nenhum elemento tinha ideia de como realizar este tipo de funcionalidade preliminarmente. Porém, observando o trabalho desenvolvido consideramos ter cumprido com os objetivos do trabalho proposto.

Como possíveis melhorias, olhámos para a arquitetura, onde poderíamos transitar da arquitetura monolítica atual para uma arquitetura mais próxima de micro-serviços. Relativamente à aplicação, mais funcionalidades poderiam ser implementadas ou melhoradas, como a existência de mais tipos de recursos (e permitir os utilizadores adicionarem novos tipos de recursos), comentário com partilha de conteúdo ou partilha dos recursos pelas diversas redes sociais existentes.

De forma a concluir e reiterar, a elaboração deste projeto permitiu a todos os constituintes do grupo alargar os seus conhecimentos relativamente a conceitos elaborados em contexto de aula e extra-aula.

A Definição do modelo de dados associado aos utilizadores

```
{
  "_id": ObjectId,
  "username": "string",
  "first_name": "string",
  "last_name": "string",
  "password": "string (hash + salt)",
  "email": "string",
  "is_admin": boolean,
  "is_active": boolean,
  "filiation": {
    "institution": "string",
    "position": "string"
  },
  "notifications": [
  ]
}
```

Código 3: Definição do modelo de dados associado aos utilizadores

B Definição do modelo de dados associado aos recursos

```
{
  "_id": ObjectId,
  "path": "string",
  "name": "string",
  "mime_type": "string",
  "image": "string",
  "type": "string",
  "description": "string",
  "author": {
    "_id": ObjectId,
    "name": "string"
  },
  "year": Number,
  "size": Number,
  "date_added": Date,
  "subject": "string",
  "tags": [
    "string",
    "string",
    ...
  ],
  "comments": [
    {
      "author": {
        "_id": ObjectId,
        "name": "string"
      },
      "description": "string",
      "date": Date,
      "comments": [
        {
```

```

        "author": {
            "_id": ObjectId,
            "name": "string"
        },
        "description": "string",
        "date": Date
    },
    ...
]
},
...
],
"rating": {
    "score": Number,
    "votes": Number
}
}

```

Código 4: Definição do modelo de dados associado aos recursos

C Função de autenticação Local

```
passport.use(  
  new LocalStrategy(  
    {  
      usernameField: "username",  
      passwordField: "password",  
    },  
    (username, password, done) => {  
      axios  
        .post("auth/login", { username: username, password: password })  
        .then((dados) => {  
          const user = dados.data;  
          if (!user) {  
            return done(null, false,  
              { message: "Utilizador inexistente!\n" });  
          }  
          return done(null, user);  
        })  
        .catch((erro) => {  
          done(erro);  
        });  
    },  
  ),  
);
```

Código 5: Função de autenticação Local

D Função de autenticação *JWT*

```
passport.use(  
  new JWTStrategy(  
    {  
      jwtFromRequest: (req) => req.cookies.token,  
      secretOrKey: process.env.JWT_SECRET_KEY,  
    },  
    (jwtPayload, done) => {  
      if (Date.now() > jwtPayload.expires) {  
        return done("jwt expired");  
      }  
      axios.get("users/" + jwtPayload.username)  
        .then((dados) => {  
          if (dados.data && dados.data.is_active && dados.data.token  
            && dados.data.token !== "") {  
            return done(null, dados.data);  
          } else {  
            return done(null, false);  
          }  
        })  
        .catch((erro) => {  
          return done(erro, false);  
        });  
    },  
  ),  
);
```

Código 6: Função de autenticação *JWT*

E Função de autenticação *Google*

```
passport.use(  
  new GoogleStrategy(  
    {  
      clientID: "<CLIENT_ID>",  
      clientSecret: "<CLIENT_SECRET>",  
      callbackURL: "<CALLBACK_URL>",  
      proxy: true,  
    },  
    function (accessToken, refreshToken, params, profile, done) {  
      axios.get("users/byEmail?email=" + profile._json.email)  
        .then((dados) => {  
          if (dados.data != null) {  
            dados.data.accessToken = accessToken;  
            axios.put("auth/updateAccessToken", { dados: dados.data })  
              .then((user) => {  
                done(null, user.data);  
              })  
              .catch((erro) => done(erro, false));  
          } else done(profile._json, false);  
        })  
        .catch((erro) => done(erro, false));  
    },  
  ),  
);
```

Código 7: Função de autenticação *Google*

F Função de validação '*BagIt*'

```
function bagItConventions(files) {
  let filePath = files.filter((file) =>
    file.type !== "directory").map((file) => file.path);

  // check for manifest file
  if (!filePath.includes("manifest.json")) {
    return false;
  }

  // check if all data is under "data/" folder
  if (filePath.filter((path) => !"/^data\\/".test(path)).length !== 1) {
    return false;
  }

  // check if all files in manifest are the same in the package
  let manifest = JSON.parse(files[filePath.map((file) =>
    file.path).indexOf("manifest.json")].data.toString());
  filePath.splice(filePath.indexOf("manifest.json"), 1);

  if (!(JSON.stringify(manifest.files.sort()) ===
    JSON.stringify(filePath.sort()))) {
    return false;
  }

  return true;
}
```

Código 8: Função de validação '*BagIt*'

G Excerto do mecanismo de armazenamento dos recursos

```
(...)  
  
let pathFolder = "uploads/" + fields.type + "/" +  
    user.username + "/" + new Date().getTime();  
  
(...)  
  
function storeResource(files, pathFolder) {  
    files.forEach((file) => {  
        if (file.type !== "directory") {  
            fsPath.writeFile("app/" + pathFolder + "/content/" + file.path,  
                file.data,  
                function (err) {  
                    if (err) {  
                        console.error(err);  
                    }  
                });  
        }  
    });  
}
```

Código 9: Excerto do mecanismo de armazenamento dos recursos

H Pipeline de CI/CD no *GitHub*

```
name: CI

on:
  push:
    branches: [master]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Create env file
        env:
          JWT_SECRET_KEY: ${ secrets.JWT_SECRET_KEY }
          MONGODB_URL_ATLAS: ${ secrets.MONGODB_URL_ATLAS }
        run: |
          touch .env
          echo JWT_SECRET_KEY=${JWT_SECRET_KEY} >> .env
          echo JWT_SECRET_TIME="36000" >> .env
          echo API_URL="https://unifiles.herokuapp.com/api" >> .env
          echo MONGODB_URL=${MONGODB_URL_ATLAS} >> .env
          echo MONGODB_URL_ATLAS=${MONGODB_URL_ATLAS} >> .env
          cat .env

      - name: Deploy to Heroku
        uses: akhileshns/heroku-deploy@v3.8.9
        with:
          heroku_api_key: ${ secrets.HEROKU_API_KEY }
          heroku_app_name: ${ secrets.HEROKU_APP_NAME }
          heroku_email: ${ secrets.HEROKU_EMAIL }
```

Código 10: Pipeline de CI/CD no *GitHub*

Referências

- [1] *RFC 8493 - The BagIt File Packaging Format (V1.0)*, "<https://tools.ietf.org/html/rfc8493>", Acedido: 01-01-2021.
- [2] *Install MongoDB - MongoDB Manual*, "<https://docs.mongodb.com/manual/installation>", Acedido: 05-02-2021.