

Trabalho Prático - Plataforma de Gestão e Disponibilização de Recursos Educativos

**Desenvolvimento de Aplicações Web
Processamento e Representação de Informação
Relatório de Desenvolvimento**

Ana Catarina Gil- A85266
Hugo Cunha - A84656
Tânia Rocha - A85176

7 de fevereiro de 2021

Resumo

Trabalho realizado no âmbito das unidades curriculares Processamento e Representação de Informação e Desenvolvimento de Aplicações Web do 4º ano do Mestrado Integrado em Engenharia Informática.

O projeto consiste no desenvolvimento de uma Aplicação Web que implemente um repositório digital de recursos educativos.

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

Conteúdo

| | | |
|----------|---|-----------|
| 1 | Introdução | 3 |
| 2 | Aplicação Web | 4 |
| 2.1 | Arquitetura OAIS | 4 |
| 2.2 | Estrutura | 5 |
| 2.3 | Funcionalidades | 5 |
| 2.3.1 | Administrador | 6 |
| 2.3.2 | Produtor | 6 |
| 2.3.3 | Consumidor | 6 |
| 2.4 | Dependências | 7 |
| 3 | Modelação | 7 |
| 3.1 | Models | 7 |
| 3.2 | Controllers | 8 |
| 4 | Armazenamento de Recursos | 9 |
| 4.1 | SIP por processo de Ingestão | 9 |
| 4.2 | AIP e o armazenamento de Recursos | 10 |
| 4.3 | DIP e o Download de Recursos | 10 |
| 5 | Testes e Resultados | 10 |
| 6 | Conclusões | 14 |

Lista de Figuras

| | | |
|---|--|----|
| 1 | Arquitetura OAIS. | 4 |
| 2 | Editar o Perfil de um Utilizador. | 10 |
| 3 | Página de Login. | 11 |
| 4 | Página CRUD dos anúncios. | 11 |
| 5 | Página inicial de um Administrador. | 12 |
| 6 | Página CRUD de recursos de um Administrador. | 12 |
| 7 | Página inicial de um Produtor. | 13 |
| 8 | Página para efetuar carregamento de ficheiros. | 13 |

1 Introdução

No actual projeto foi proposto o desenvolvimento de uma *Web Application* com o intuito de gerir e armazenar recursos educativos seguindo a arquitectura reconhecida OAIS (Online Archive Information System). A modelação desenvolvida foi estruturada com o objetivo de permitir pesquisas simples e interactivas ou criação e gestão de recursos que se apresentam em formato digital.

Para tal, foi utilizada a ferramenta NodeJS para o desenvolvimento e o MongoDB como base de dados onde são armazenados os dados.

2 Aplicação Web

2.1 Arquitetura OAIS

A arquitectura OAIS, *Open Archive Information System*, é considerado como um *archive* consistido por utilizadores e sistemas que tem como objetivo preservar informação em *long term* e torna-la disponível para uma determinada comunidade.

O ambiente de desenvolvimento desta arquitectura envolve 3 entidades principais:

- Produtores de informação;
- Consumidores de informação;
- Administrador.

Em termos funcionais, o sistema é constituído por 3 processos principais:

- Ingestão - Recepção/deposito de informação;
- Administração - Processo responsável pela gestão interna do sistema: gestão dos objectos arquivados, gestão de utilizadores, etc;
- Disseminação - Processo responsável pela disseminação/distribuição/publicação dos objectos arquivados tanto a partir de ZIPs com a informação ou através da visualização da mesma num website construído para tal.

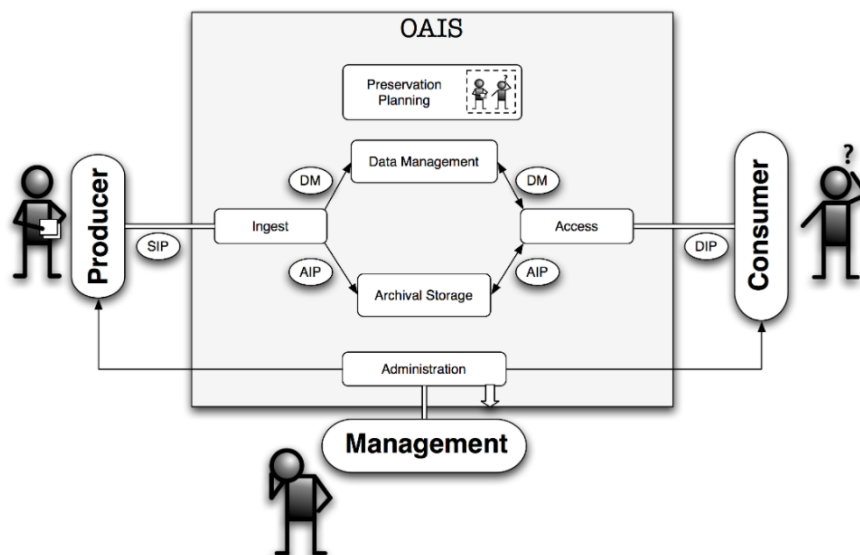


Figura 1: Arquitectura OAIS.

De acordo com o OAIS e com a imagem anterior podemos observar três tipos de pacotes de informação a circular no sistema:

- SIP "*Submission Information Package*"- Pacote que é enviado pelo produtor ao sistema para ser processado e arquivado. A sua estrutura terá de ser especificada no início do projecto.
- AIP "*Archival Information Package*"- Pacote arquivado, ou seja, um SIP depois de processado e armazenado torna-se num AIP. Este terá uma determinada estrutura que irá depender da forma como será armazenado.
- DIP "*Dissemination Information Package*" - Pacote oferecido ao consumidor. Tanto poderá ser um website a partir do qual aquele consiga navegar nos conteúdos como pode ser um ficheiro ZIP com um conjunto de conteúdos previamente seleccionados.

2.2 Estrutura

Para o desenvolvimento da nossa aplicação *Web* foi desenvolvida uma *API* denominada por *DAW-PRI-2020* criada através da ferramenta *Express*. Esta encontra-se dividida entre *Back-End* e *Front-End*, seguindo a estrutura de um modelo MVC. Assim a nossa aplicação é composta pelos componentes :

- **model** : Lida com a Base de Dados. É onde se encontra definido os *Schemas* das coleções presentes na base de dados do sistema implementado, sendo estas:
 - Recursos
 - Utilizadores
 - Anuncios
- **controller**: Onde se encontra as funcionalidades e a lógica da aplicação. Nestas encontram-se implementadas para cada uma das identidades correspondentes a cada model, as queries que auxiliam a aquisição das informações existentes na BD.
- **routes**: Onde se encontram definidas as rotas suportadas pela nossa aplicação. Estas encontram-se divididas de acordo com os três tipos de utilizadores e com um geral no caso de nenhuma sessão estar iniciada.
- **views**: Onde se encontram definidas as páginas apresentadas aos utilizadores no *browser*.
- **public**: Esta pasta contém:
 - *javaScripts*: contém scripts que auxiliam a implementação de *views*
 - *fileStore*: pasta onde são armazenados os ficheiros associados a cada recurso
 - *images*: imagens utilizadas para o desenvolvimento das *views*

2.3 Funcionalidades

Para permitir uma diversidade de funcionalidades sobre dados de perspectivas e privilégios diferentes foram desenvolvidos três níveis na aplicação que se enquadram nos requisitos da arquitectura OAIS que se pretende seguir:

- Administrador;
- Produtor;
- Consumidor.

2.3.1 Administrador

O Administrador segue o modelo Utilizador e é o que tem o maior nível de privilégios e acessos sobre recursos e outros utilizadores.

Foi implementado com o objetivo de realizar as tarefas **CRUD**, isto é Create, Read, Update and Delete, sobre recursos, utilizadores e anúncios.

Dentro destas tarefas são destacadas:

- Listagem e consulta de todos os utilizadores do sistema;
- Registo, remoção e atualização ou edição de outros utilizadores;
- Listagem, consulta e download de todos os recursos do sistema;
- Registo, remoção e atualização ou edição de recursos e remoção dos respetivos comentários caso seja necessário;
- Pontuar e comentar os recursos;
- Listagem de todos os anúncios do sistema criados por todos os administradores;
- Registo, remoção e activação ou desactivação dos anúncios;
- Pode utilizar a aplicação como se fosse outro tipo de utilizador o que permite que possa sempre verificar como está o funcionamento da aplicação noutros contextos.

2.3.2 Produtor

No nível mais abaixo de privilégios temos o utilizador do tipo Produtor. Estes utilizadores tem varias funcionalidades sobre dados relativos ao mesmo ou aos seus recursos, tais como:

- Listagem, consulta e download de todos os recursos públicos e privados da sua autoria;
- Registo, remoção e edição dos seus próprios recursos;
- Pontuar e comentar os recursos;
- Edição de dados pessoais.

2.3.3 Consumidor

Por fim o utilizador de nível de privilégios mais baixo do sistema, o Consumidor, que tem apenas as tarefas:

- Listagem, consulta e download de recursos públicos;
- Pontuar e comentar os mesmos recursos;
- Alteração de dados pessoais.

2.4 Dependências

Durante a implementação foram utilizados módulos ou pacotes NPM que permitiram variadas tarefas. O ficheiro *package.json* contém as informações sobre os módulos utilizados. Este ficheiro inclui para cada módulo a sua versão, nome, descrição e dependências associadas.

Algumas dependências de particular importância são:

- *mongoose* - Para comunicação com a base de dados MongoDB.;
- *passport* - Permite tratar de processos relacionados com a autenticação no sistema;
- *adm-zip* - Permite efetuar acções de *zip* e *unzip* de ficheiros.

3 Modelação

3.1 Models

Tal como referido anteriormente, para o desenvolvimento da nossa aplicação Web foram definidos três tipos de models, estando estes armazenados na Base de Dados.

- **Model do Recurso**

O modelo do recurso é o modelo mais complexo do sistema. Além de possuir os atributos que achamos necessários este tem para a sua caracterização, eles também possuem um array de comentários, sendo que estes podem ter associados aos mesmos outros comentários, funcionando como replies de um comentário principal.

```
tipo: String,
descricao: String,
titulo: String,
dataCriacao: Date,
dataRegisto: Date,
visibilidade: String,
numDowns: Number,
autor: [String],
produtor: String,
pontuacao: Number,
numPontuacoes: Number,
comentarios: [comentarios],
tags: [String]
```

- **Sub-Schema comentarios:**

```
_id: Number,
```



```
id_coment: Number,  
id_utilizador: String,  
nome_utilizador: String,  
data: Date,  
descricao: String  
  
comentarios.add({ comentarios: [comentarios] });
```

- **Model do Utilizador**

O modelo do utilizador é mais simples e contém a informação relativa ao mesmo.

```
nome: String,  
email: String,  
filiacao: String,  
nivel: String,  
dataRegisto: Date,  
dataUltimoAcesso: Date,  
password: String
```

- **Model do Anuncio**

Este modelo serve para armazenar os anúncios criados por qualquer administrador apresentados posteriormente nas views dos utilizadores.

```
autor: String,  
dataCriacao: Date,  
ativo: Number,  
anuncio: String
```

3.2 Controllers

Depois de definirmos todos os modelos de dados, implementamos os respetivos controllers, permitindo efetuar ações básicas sobre a base de dados, tais como:

- **Consultar:**

Exemplo: Listagem de todos os utilizadores do sistema

```
module.exports.list = function () {  
  return Utilizador.find().exec();  
};
```

- **Inserir:**

Exemplo: Inserção de um novo utilizador

```
module.exports.insert = function (u) {  
  var novoUtilizador = new Utilizador(u);  
  return novoUtilizador.save();  
};
```

- **Eliminar:**

Exemplo: Eliminar utilizador através do seu id

```
module.exports.remove = function (id) {  
  return Utilizador.deleteOne({ _id: id });  
};
```

- **Atualizar:**

Exemplo: Atualiza dados do utilizador com um determinado id

```
module.exports.edit = function (id, u) {  
  return Utilizador.findByIdAndUpdate(id, u, { new: true });  
};
```

Apesar de todos os exemplos apresentados serem referentes ao model Utilizador, os mesmos foram também implementados para os restantes models.

Como seria de esperar foram criadas mais queries do que as apresentadas nesta secção de forma a nos auxiliar nas diversas ações implementadas.

4 Armazenamento de Recursos

Para armazenar os recursos usamos o sistema de ficheiros onde corre a API. Cada recurso é armazenado numa estrutura semelhante a BagIt.

4.1 SIP por processo de Ingestão

Inicialmente é requisitado um ficheiro ZIP ao utilizador aquando o upload de conteúdo que será o nosso pacote SIP.

O processo de ingestão passa por validar cada ficheiro contra um manifesto previamente definido guardado na raiz da aplicação. São executadas algumas validações para garantir que não há riscos de danificar sistemas tipo DOS como por exemplo uma *blacklist* de nomes e de caracteres dos nomes de ficheiros, uma *whitelist* de extensões permitidas e um tamanho máximo para o SIP.

É neste momento que os metadados do ficheiro são guardados em base de dados fornecendo um id correspondente ao futuro AIP.

4.2 AIP e o armazenamento de Recursos

Antes de terminar o processo de Ingestão, é calculado um hash com o algoritmo SHA-256 para cada subficheiro do SIP. Posteriormente é guardado num manifest o hash correspondente a cada ficheiro do mesmo SIP. Este ficheiro, por sua vez é transformado num AIP, é gerada uma pasta para representar este AIP e, segundo a especificação, é criado um terceiro ficheiro bag.txt que para além de indicar o path corrente de um AIP contém também o nome do ficheiro ZIP guardado, para facilitar o acesso.

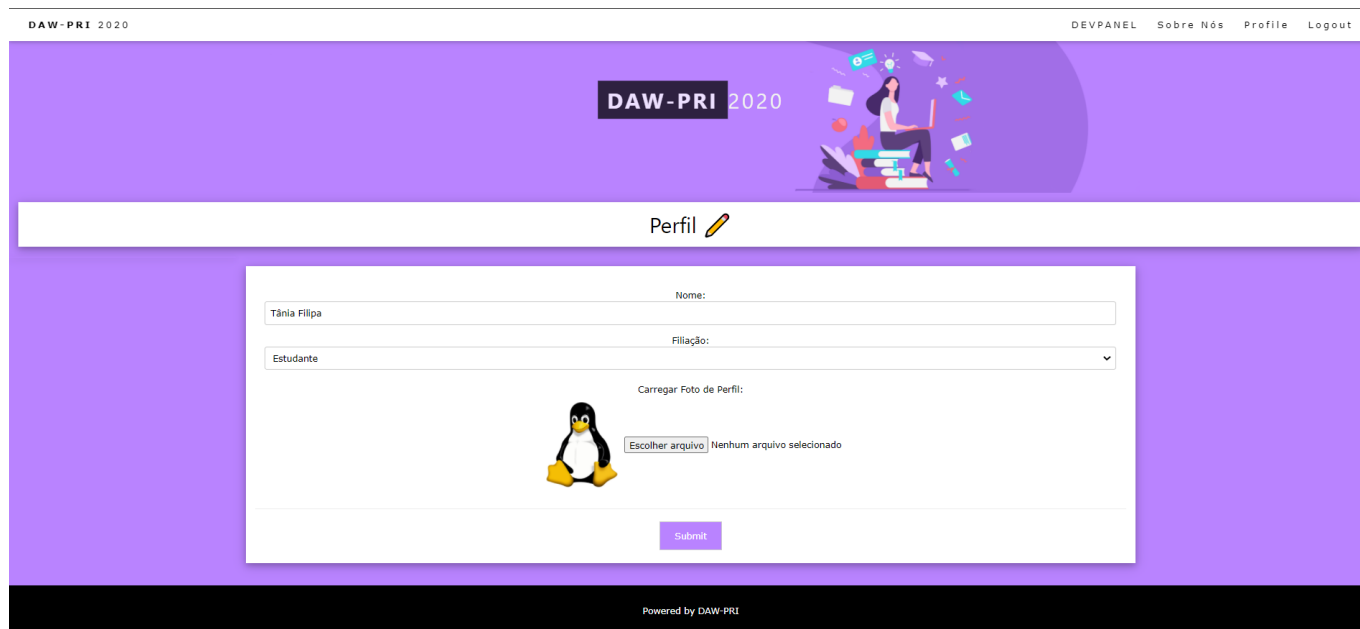
4.3 DIP e o Download de Recursos

De cada vez que é pretendido distribuir o ficheiro, é feita uma consulta inicial á base de dados para obter o ID do mesmo, assim com esta chave procura-se na pasta FileStore do arquivo do sistema, o ficheiro correspondente. Depois de confirmar a integridade do AIP, fazendo o matching dos hashes, é retirado o ZIP com os dados do AIP correspondente, enviando o mesmo para um cliente que pretenda fazer o download.

Também é possível extrair-lo localmente o que permite apresentar uma preview na página do mesmo recurso.

5 Testes e Resultados

Para permitir uma pré visualização dos resultados e confirmar que tudo decorre como planeado, foram feitos alguns testes e consultas no sistema. Podem ser observados em baixo:



The screenshot shows the 'Perfil' (Profile) editing page of the DAW-PRI 2020 system. The page has a purple header with the 'DAW-PRI 2020' logo and a navigation bar with links: 'DEVPANEL', 'Sobre Nós', 'Profile', and 'Logout'. Below the header, there's a 'Perfil' section with a pencil icon. The main content area is a form for editing the profile. It includes a 'Nome:' field with the value 'Tânia Filipa', a 'Filiação:' dropdown menu with 'Estudante' selected, and a 'Carregar Foto de Perfil:' section. The photo section features a penguin icon, a file selection button labeled 'Escolher arquivo', and the text 'Nenhum arquivo selecionado'. A 'Submit' button is at the bottom of the form. The footer of the page says 'Powered by DAW-PRI'.

Figura 2: Editar o Perfil de um Utilizador.

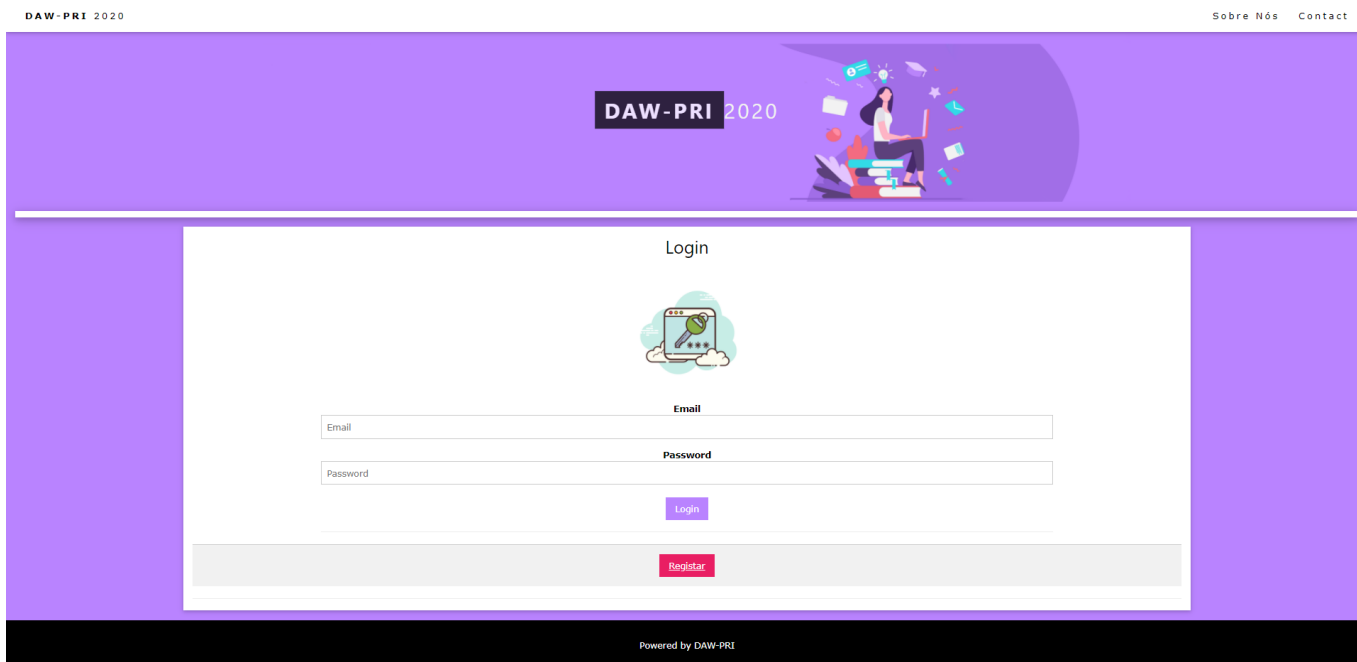


Figura 3: Página de Login.

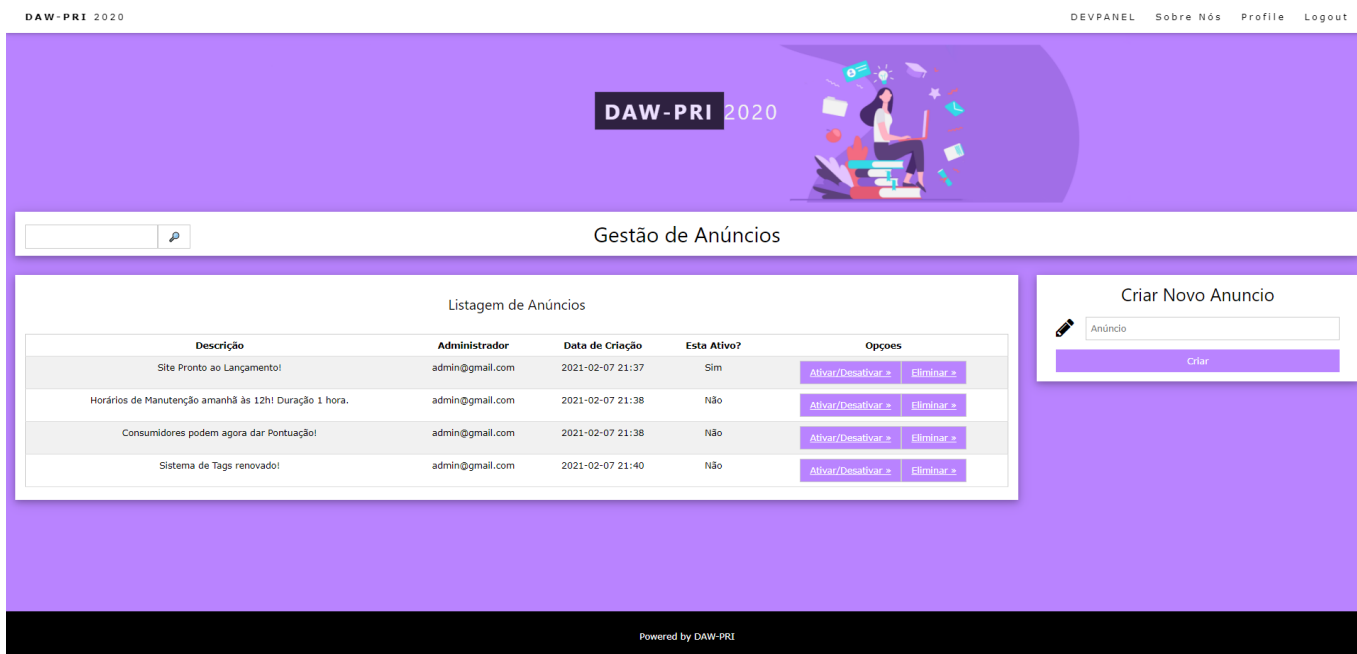


Figura 4: Página CRUD dos anúncios.

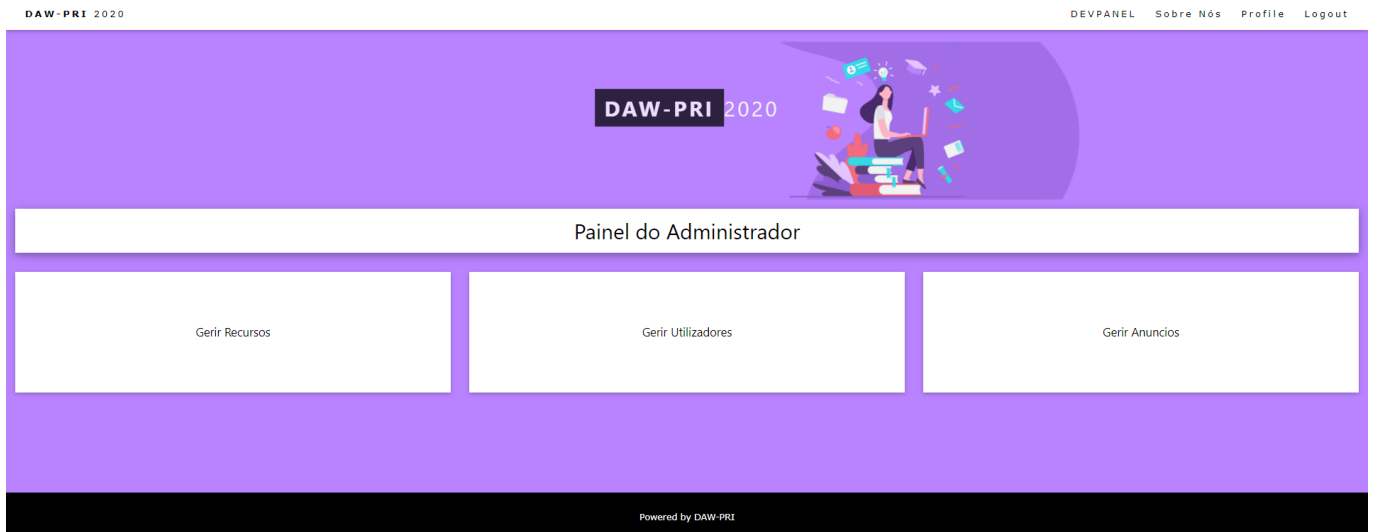


Figura 5: Página inicial de um Administrador.

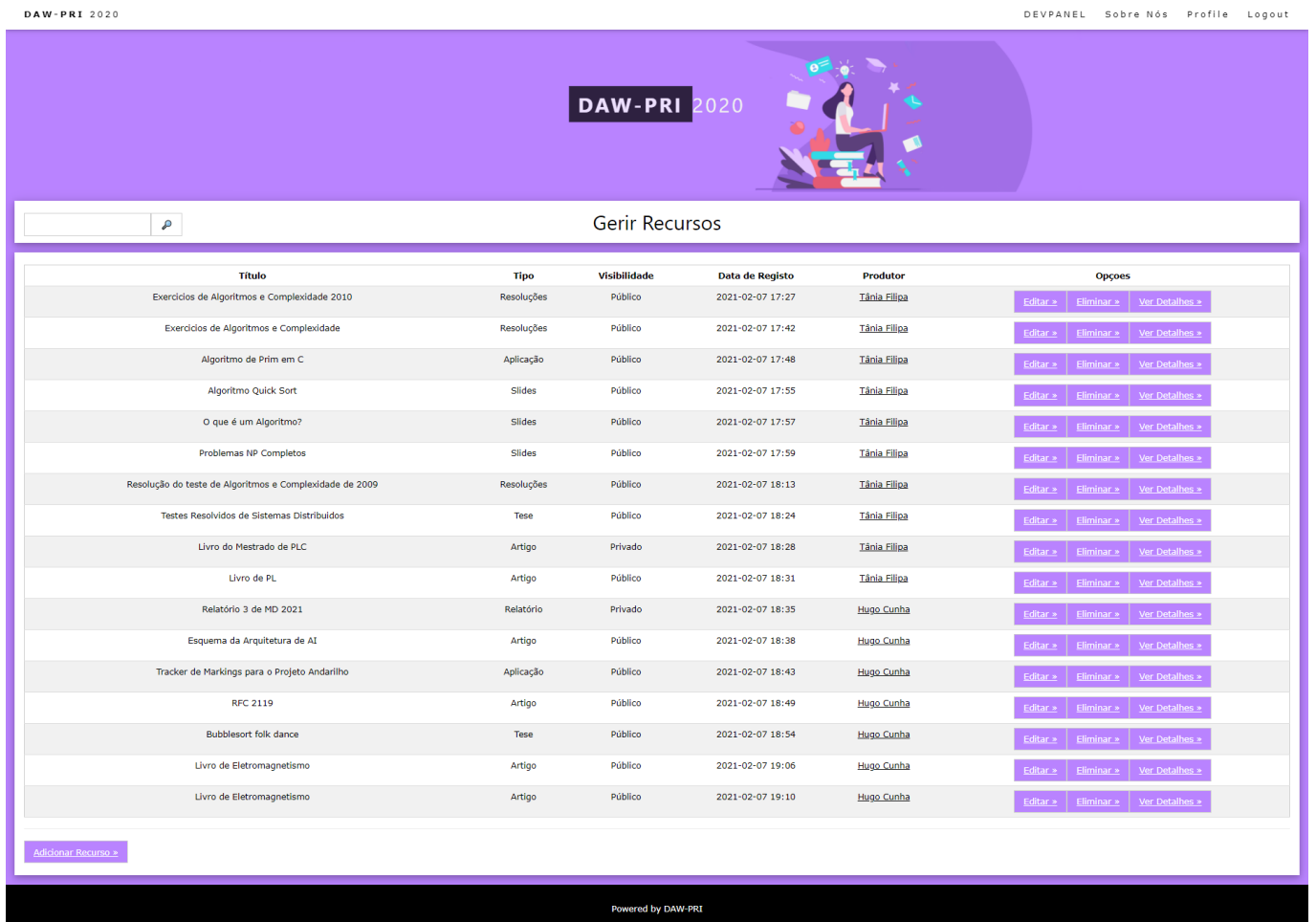


Figura 6: Página CRUD de recursos de um Administrador.

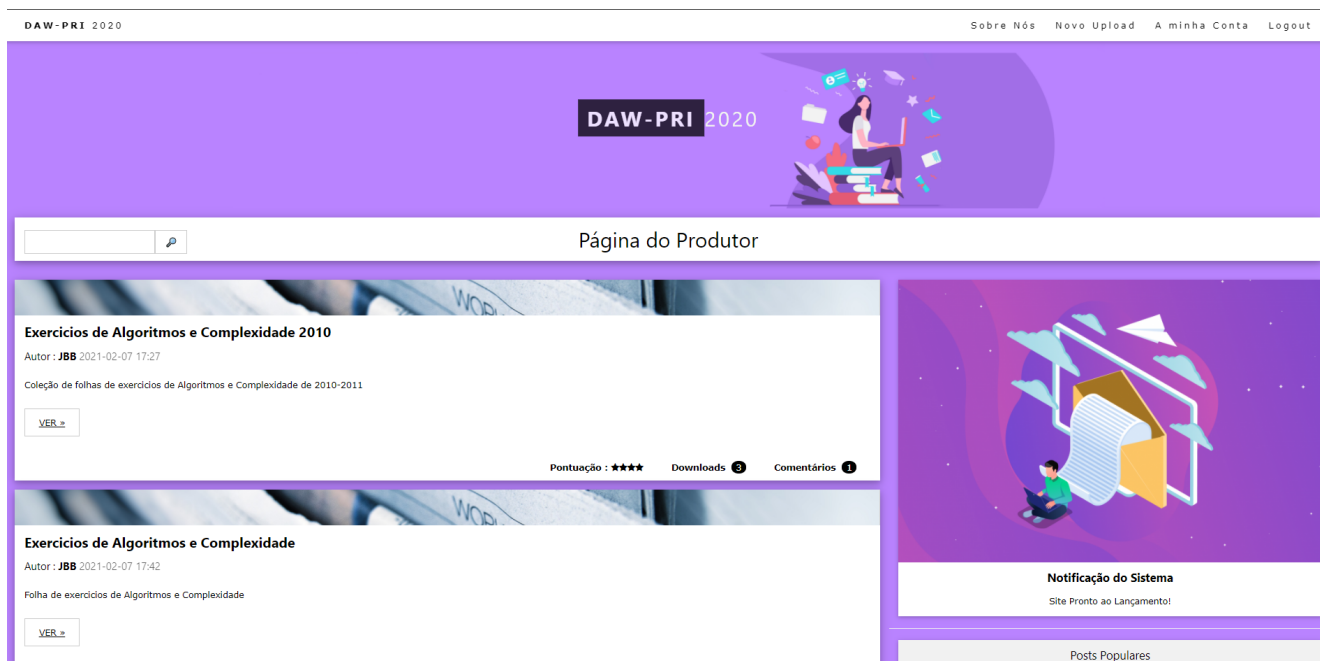


Figura 7: Página inicial de um Produtor.

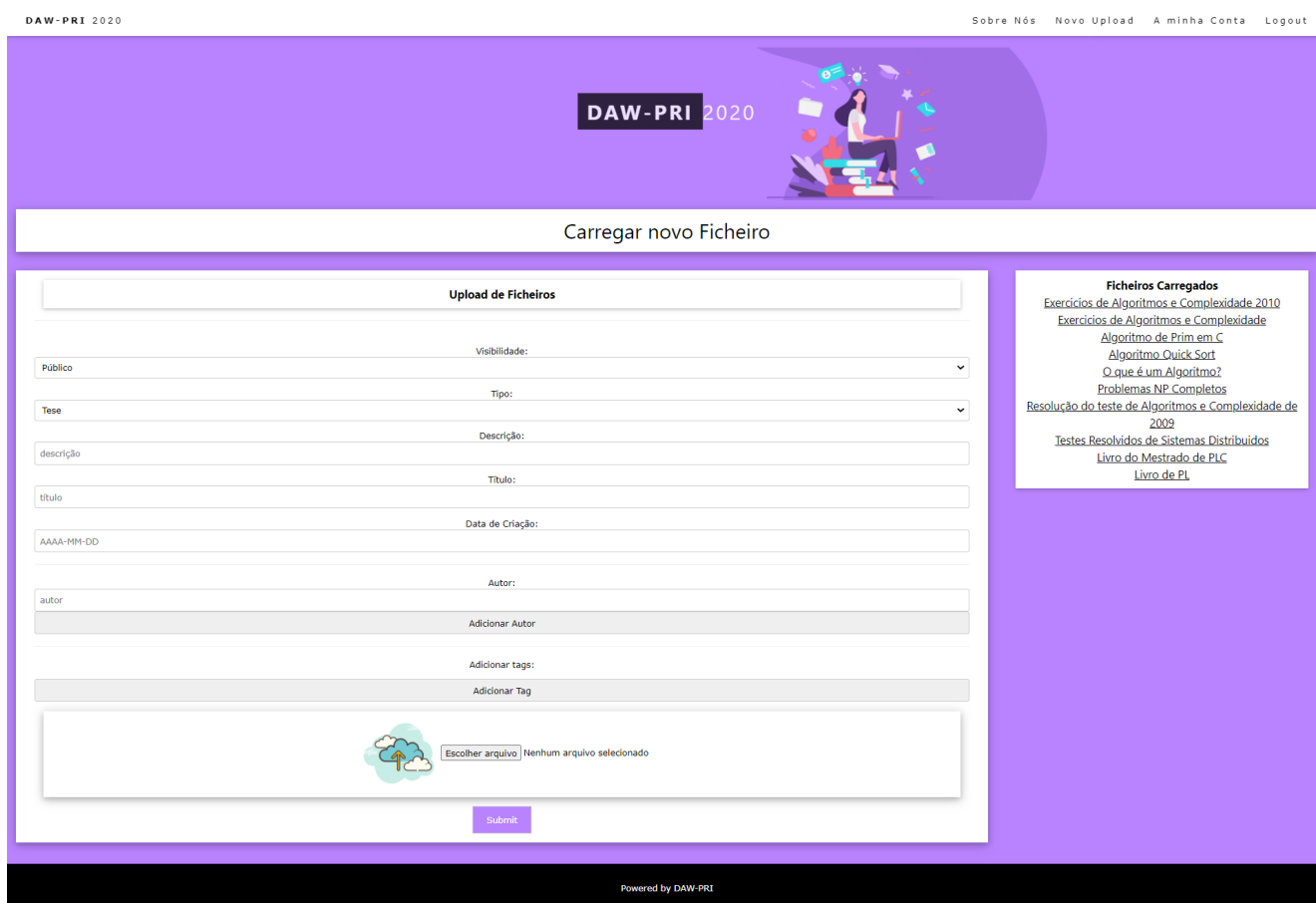


Figura 8: Página para efetuar carregamento de ficheiros.

6 Conclusões

Com o desenvolvimento deste projeto, que tinha como objetivo desenvolver uma plataforma de gestão e disponibilização de recursos educativos, conseguimos pôr em prática e aprofundar o conteúdo lecionado nesta UC ao longo do semestre.

De forma geral cumprimos todos os requisitos propostos no enunciado. No entanto, dada a natureza e o tema deste projeto, poderíamos adicionar novas funcionalidades ao mesmo de forma a torná-lo mais realista e pronto para liberá-lo no mercado. Assim como trabalho futuro poderíamos implementar o início de sessão com outras aplicações, por exemplo *google*, suportar maiores dimensões e outras extensões de ficheiros, tal como a preview dessas mesmas extensões.

A principal dificuldade sentida ao longo dos trabalho foi a manipulação dos ficheiros *ZIP*, porém, a gestão do tempo também foi desafiante, visto que gostaríamos de ter evoluído um pouco mais o trabalho, tal como descrito anteriormente.

Posto isto, achamos que implementamos todas as ferramentas e funcionalidades necessárias para um bom comportamento e performance da nossa aplicação *Web*.