Universidade do Minho
Mestrado Integrado em Engenharia Informática

Representação e Processamento de Conhecimento na Web

André Martins, A84347 Filipe Fernandes, A83996
Inês Marinho, PG47625

Braga, 19 de Junho de 2022

Conteúdo

1 Introdução

Neste projeto pretende-se a criação de um repositório para recursos didáticos, que irá armazenar vários tipos de conteúdo. Estes serão publicados por um tipo de utilizador a quem se chama de *producer*, além deste, irá existir um *consumer* que poderá usufruir dos conteúdos e um *admin* que terá acesso a tudo, de modo a tornar mais fácil a gestão de todo o repositório.

2 Desenvolvimento

O sistema foi divido em quatro "blocos": Base de dados, API de Autenticação, API de dados e APP server.

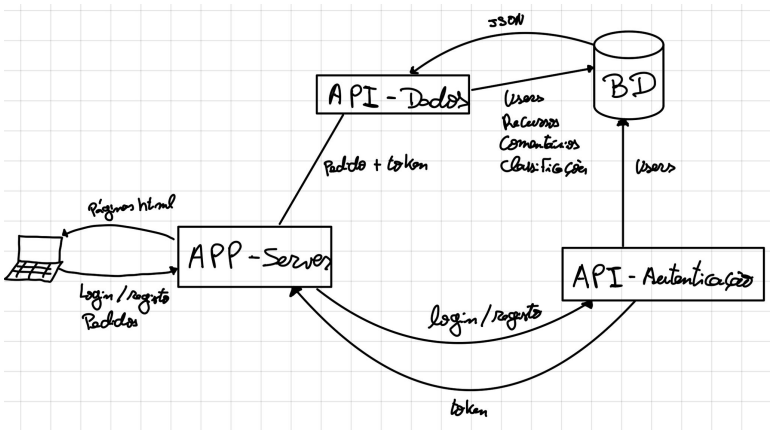

Figura 1: Representação simplificada do sistema

2.1 Base de Dados

A base de dados utilizada foi o MongoDB, como solicitado no enunciado do projeto. Foi então criada a base de dados RRD com as seguintes *collections*:

- Users
- Recurso
- Comments
- Classifications

2.1.1 Estrutura dos documentos nas *collections*

- Users

```
username: String,
email: String,
level: String,
image: String,
descricao: String,
followers : [String] //foi utlizado o termo errado, não é
//followers mas sim following
```

- Recurso

```
title: String,
author: String,
user: String,
data: String,
tipo: String,
public: Boolean,
deleted: Boolean,  //Recurso apagado==true
deleteDate : String,  //data que o recurso foi apagado
deleteUser : String,  //utilizador que apagou o recurso
path: String,
```

- Comments

```
recurso: String,
user: String,
data: String,
deleted: Boolean,  //Comentário apagado==true
deleteDate : String,  //data que o comentário foi apagado
deleteUser : String, //utilizador que apagou o comentário
comentario: String
```

- Classifications

```
classificacao: Number,
numClassif: Number,  //# de users que classificaram o recurso
usersQclassific: [], //lista de users que já classificou
recurso: String      //ID do recurso
```

2.2 API de autenticação

Para o acesso aos recursos da API de dados é necessário um *token* que pode ser obtido através da realização de *login* ou registo, pois após o registo é logo gerado o *token*, não sendo necessário realizar o *login*.

Para a autenticação necessita-se do *username* e da *password* e é utilizado o *middleware* `Passport`. Para a geração do *token* é utilizado o *jwt*, com o id, level e *username* do utilizador.

Os *endpoints* de acesso à API são:

- POST /registo : criar um novo utilizador, sendo obrigatório *username*, *password*, *email* e *level*. E devolve como resposta o *token*;

- POST /login : *login* de um utilizador, que devolve como resposta o *token*.

2.3 API de dados

Para a realização de pedidos à API é necessário ter um *token* de acesso, sendo este obtido através da API de Autenticação. Depois de ter o *token* basta incluir este no *body* ou na *query string*(?token=) nos pedidos à API.

- *Endpoints* para acesso a dados relacionados com os utilizadores(/users):
  - GET / : Todos os utilizadores da base de dados e que podem ser filtrados por nível de acesso ou por nome utilizando a *query string*.
  - POST /atualizaDescricao/:id : É recebido o id do utilizador, se for o próprio ou caso tenha permissões de *admin*, poderá prosseguir com a alteração da descrição.
  - POST /atualizaImagem/:id : É recebido o id do utilizador, se for o próprio ou no caso de ter permissões de *admin*, poderá prosseguir com a alteração do *path* da imagem.
  - POST /atualizalvl/:id : Apenas um *admin* poderá usar este *endpoint* para alterar o nível de permissões de outro utilizador.
  - GET /addFollower/:id : Adiciona um *follower* com o id fornecido, ao utilizador que faz o pedido, isto é, o utilizador que faz o pedido passa a seguir o utilizador passado como parâmetro.
  - GET /getFollowers/:id : Devolve os utilizadores que o utilizador, cujo o id é passado, segue.
  - GET /getFollowing/:id : Devolve os utilizadores que o utilizador do id fornecido segue.
  - GET /unFollow/:id : É feita a remoção do *follower* do utilizador com o id fornecido.

  - GET /remove/:id : Apenas um *admin* poderá usar este *endpoint* para remoção de um utilizador.

  - GET /:id : Obtém todas as informações relativas a um utilizador.

- *Endpoints* para acesso a dados relacionados com os recursos(/recursos):

  - GET / : Apenas um *admin* poderá usar este *endpoint* para ver todas as publicações.

  - POST / : Para submeter um recurso.

  - GET /Deleted : Apenas um *admin* poderá usar este *endpoint* para ver todos os recursos eliminados.

  - GET /noDeleted : Apenas um *admin* poderá usar este *endpoint* para ver todos os recursos não eliminados.

  - GET /getMyRec : Para o utilizador ver os seus próprios recursos.

  - GET /userRecurso/:id : Para ver os recursos de um utilizador dado o seu id, caso o siga, pode ver os recursos públicos e privados, se não estiver a seguir, apenas vê os públicos. Caso seja admin pode ver todos os recursos.

  - POST /public : Para obter todos os recursos públicos, estes podem ser filtrados pelo seu título e tipo, através da *query string*.

  - POST /following : Para obter todos os recursos dos utilizadores que segue, estes podem ser filtrados pelo seu título e tipo, através da *query string*.

  - POST /alteraEstado/:id : Para alterar o estado do recurso entre público e privado, isto só pode ser feito pelo próprio ou por um *admin*.

  - POST /alteraTitulo/:id : Para alterar o título do recurso, isto só pode ser feito pelo dono do recurso ou por um *admin*.

  - POST /alteraAuthor/:id : Para alterar o autor do recurso, isto só pode ser feito pelo dono do recurso ou por um *admin*.

  - POST /classifica/:id : Dá classificação a um recurso.

  - GET /remove/:id : Para remover um recurso, isto só pode ser feito pelo próprio ou por um *admin*.

  - GET /recupera/:id : Para recuperar um recurso que tenha sido removido, só pode ser feito por quem apagou ou por um *admin*.

  - GET /:id : Para obter todas a informações relativas a um recurso, tais como o seu conteúdo, classificação e comentários, caso o recurso esteja público, ou esteja privado um user que não seja o dono, ou *admin* tem de seguir o dono do recurso.

- *Endpoints* para acesso a dados relacionados com os commentários(/comments):

  - POST /:id : Para efetuar um comentário.

  - GET /recurso/:id : Para obter todos os comentários relativos a um recurso.

  - GET /remove/:id : Para remover um comentário, isto apenas pode ser feito pelo próprio comentador ou pelo *admin*.

  - GET /:id : Para procurar um comentário em específico.

2.4 SIP e Processo de ingestão

Para a definição do ponto de entrada no sistema, o SIP e o processo de ingestão que lhe está associado, o grupo aceita um ZIP, como é pedido, e este contém: ficheiros e/ou pastas, um manifesto em formato JSON (onde cada objeto representa o nome do ficheiro e o seu *hashing* para verificação de integridade) e ainda um ficheiro *bagit* com a versão e o *encoding*.

No processo de ingestão, ao receber o ZIP, verifica se o manifesto existe e se cada ficheiro está referenciado no mesmo, dando erro quando:

- o manifesto não existe;
- a *hash* do ficheiro não é a mesma;
- quando há um ficheiro na pasta, mas não está referenciada no manifesto;
- quando há um ficheiro referenciado no manifesto, mas não existe na pasta.

No processo de armazenamento, está a ser utilizado um método discutido nas aulas, no qual se faz a *hash md5* do ficheiro ZIP que o utilizador faz *upload*. Esta *hash* tem o tamanho de 32 caracteres e são divididos em 2 grupos, cada um com 16 que irão ser o nome das pastas usadas para guardar o ZIP. No final, na pasta geral dos *uploads* irá existir uma pasta com o nome do primeiro grupo de caracteres na qual existirá outra com o segundo grupo de caracteres, dentro desta então será armazenado o ZIP e uma pasta com o nome de *unziped* onde estarão os ficheiros *pdf, xml, jpg e png* para maior facilidade na visualização do ficheiro no *browser*.

Todo o processo de *upload* e *download* dos ficheiros ZIP é feito com apoio do módulo *Multer*. Todos os ficheiros envolvidos são armazenados na pasta *fileSystem* presente no diretório principal do APP_SERVER.

2.5 Servidor webApp principal

A webApp tem como função servir os dados aos utilizadores sobre a forma de páginas HTML. Para isto foram criadas páginas de:

- Login
- Registo
- Listar recursos Públicos
- Listar recursos de utilizadores que o utilizador segue
- Listar todos os utilizadores
- Criar um novo recurso
- Visualizar a informação de um recurso
- Visualizar a informação de um utilizador

Estas páginas tem diferentes funcionalidades implementadas em si conforme o utilizador que as usa, ou seja, o nível que este detém. Sendo as seguintes, algumas dessas funcionalidades:

- Comentar um recurso;
- Apagar um comentário realizado;
- Apagar e recuperar um recurso;
- Seguir ou deixar de seguir um utilizador;
- Classificar um recurso;
- Ver recursos apagados;

- Ver os seus seguidores e quem segue;
- Visualizar a informação de um utilizador;
- Fazer *download* de um recurso;

Existem funcionalidades que:
a sua implementação não foi concluída, como:

- Num recurso visualizar os conteúdos que têm tipos que podem ser visualizados nos *browsers* (pdf, xml,png,jpeg....);
- Filtrar recursos pelo tipo e/ou título;
- Filtrar utilizadores pelo seu nome e/ou nível;
- O utilizador poder editar os seus dados: descrição, foto perfil, pedir promoção de nível;
- O utilizador poder alterar dados de um recurso que seja dono: estado, título, tipo, autor;

ou funcionalidades que estavam planeadas, mas não chegaram a ser implementadas:

- Um *consumer* poder criar recursos, mas estes terem de ser aprovados por um *producer* ou *admin*;
- *Admin* poder promover utilizador;
- Em vez de apenas poder ser carregado um ZIP no formato `bagit`, o utilizador poder carregar os ficheiros separados e depois ser criado um `bagit` válido que é depois armazenado;
- Utilizadores receberem e aceitar/recusar pedidos para seguir;

3 Instalação e utilização

3.1 Sem *docker*

Para a instalação sem `docker`, é necessário ter o `nodeJs` e `mongoDB` no computador e fazer o *download* do projeto disponível na página `github` de todos os elementos do grupo. Seguidamente, todos os pacotes necessários podem ser instalados com o comando `npm install`. Para pôr o projeto em execução, deve ter-se o mongo ativo e correr o comando `npm start` na API_Dados e no APP_server. Para correr a autenticação usa-se o comando `node app.js`. Após todos os servidores estarem ativos, é possível aceder à *WebApp* ao utilizar um *browser* e fazer um pedido para http://localhost:7003.

3.2 Com *docker*

Com o uso do `docker` a instalação torna-se muito fácil. Com o uso da nossa configuração do `docker compose` é apenas necessário correr o comando `docker-compose up` e a instalação e execução será automática. Após este processo é possível ter acesso à nossa WebApp ao utilizar um *browser* e fazer um pedido para http://localhost:7003 .

4 Demonstração

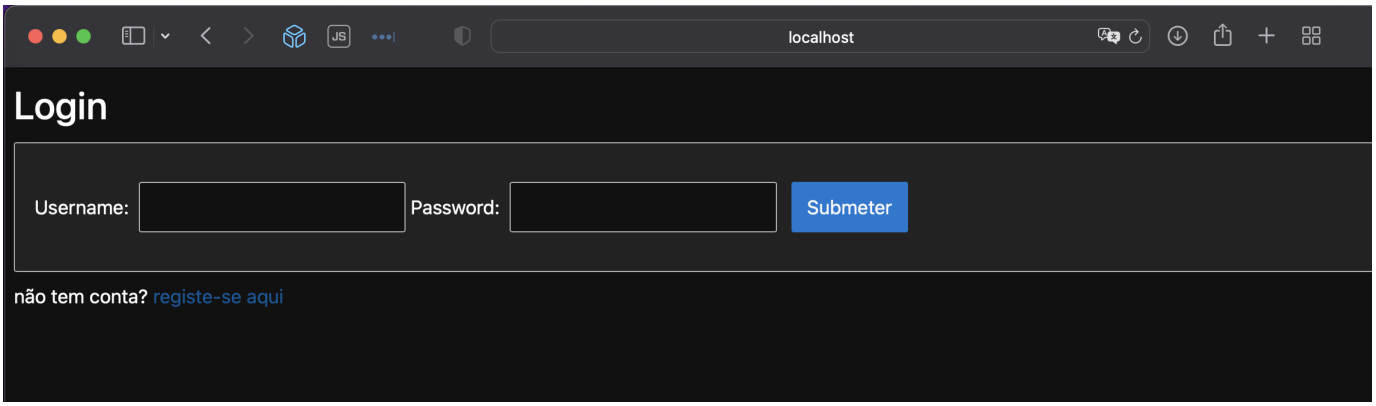

Figura 2: Página de login

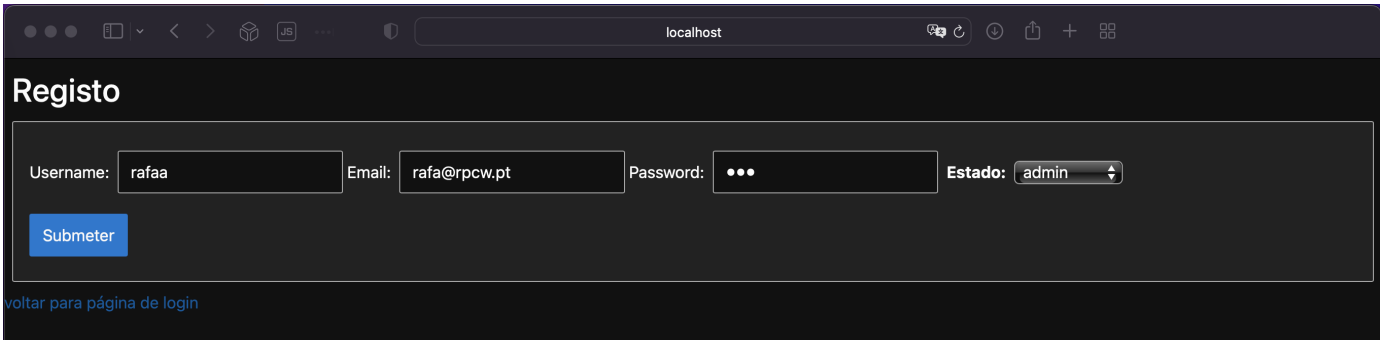

Figura 3: Página de registo

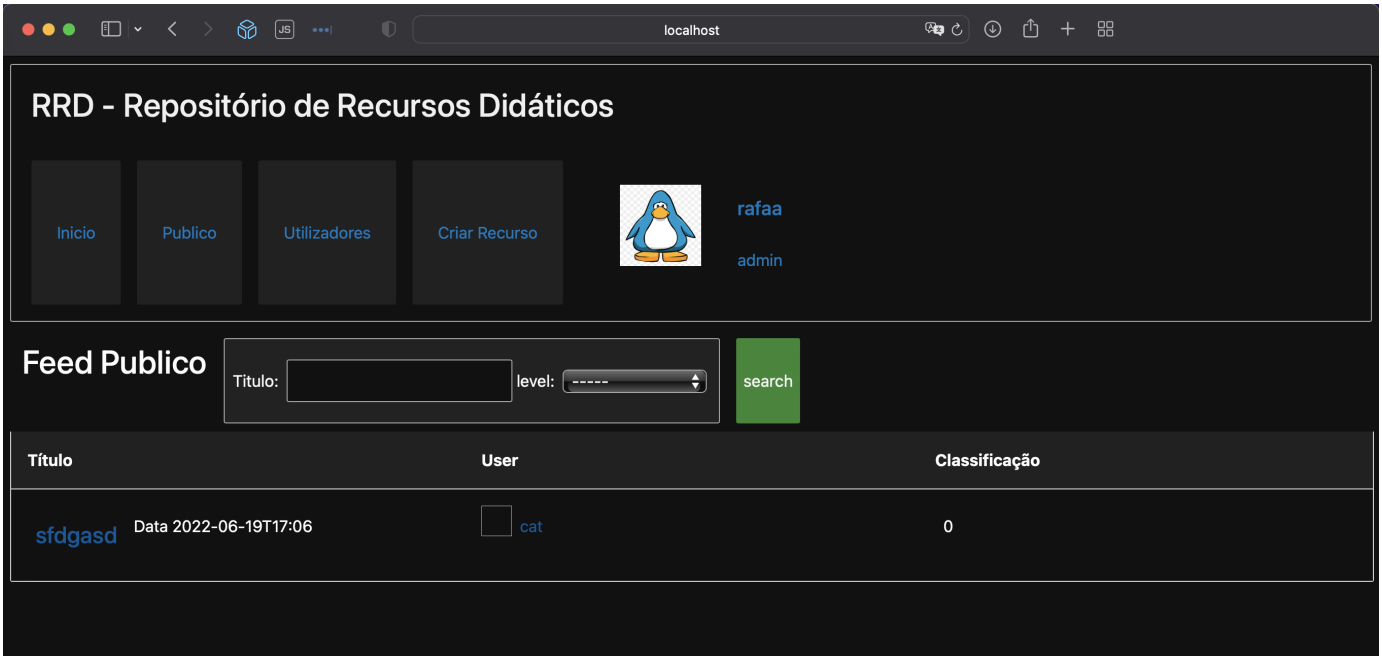

Figura 4: Utilizador com nível admin

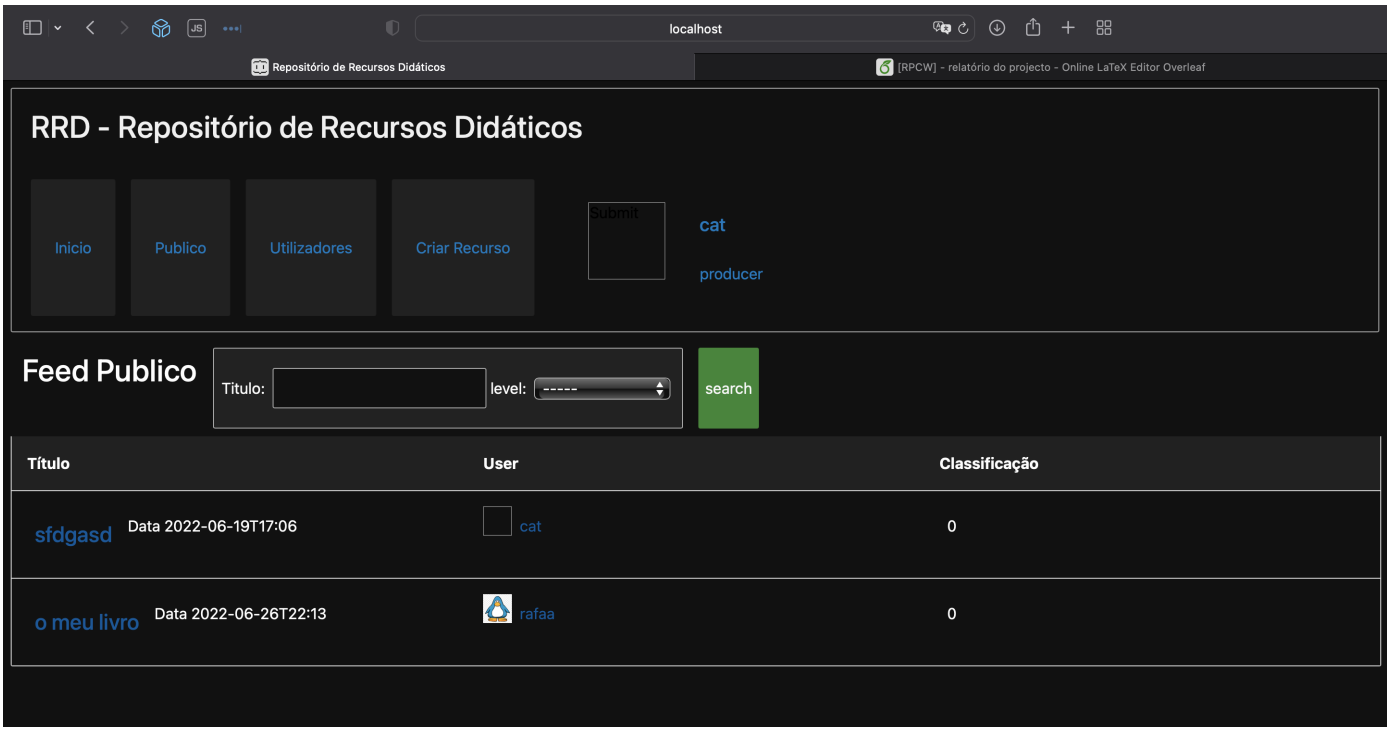

Figura 5: Utilizador com nível *producer*

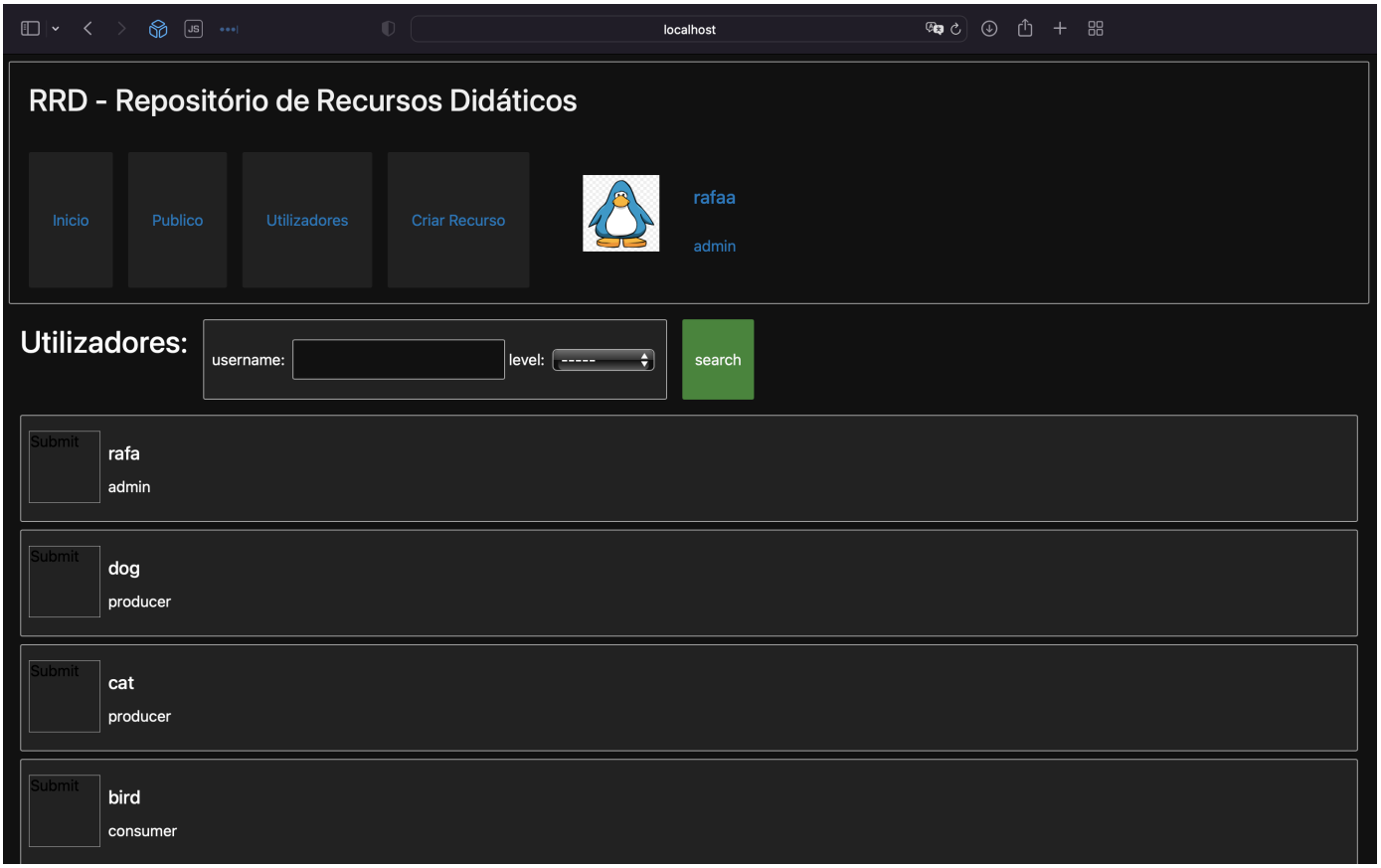

Figura 6: Página de utilizadores

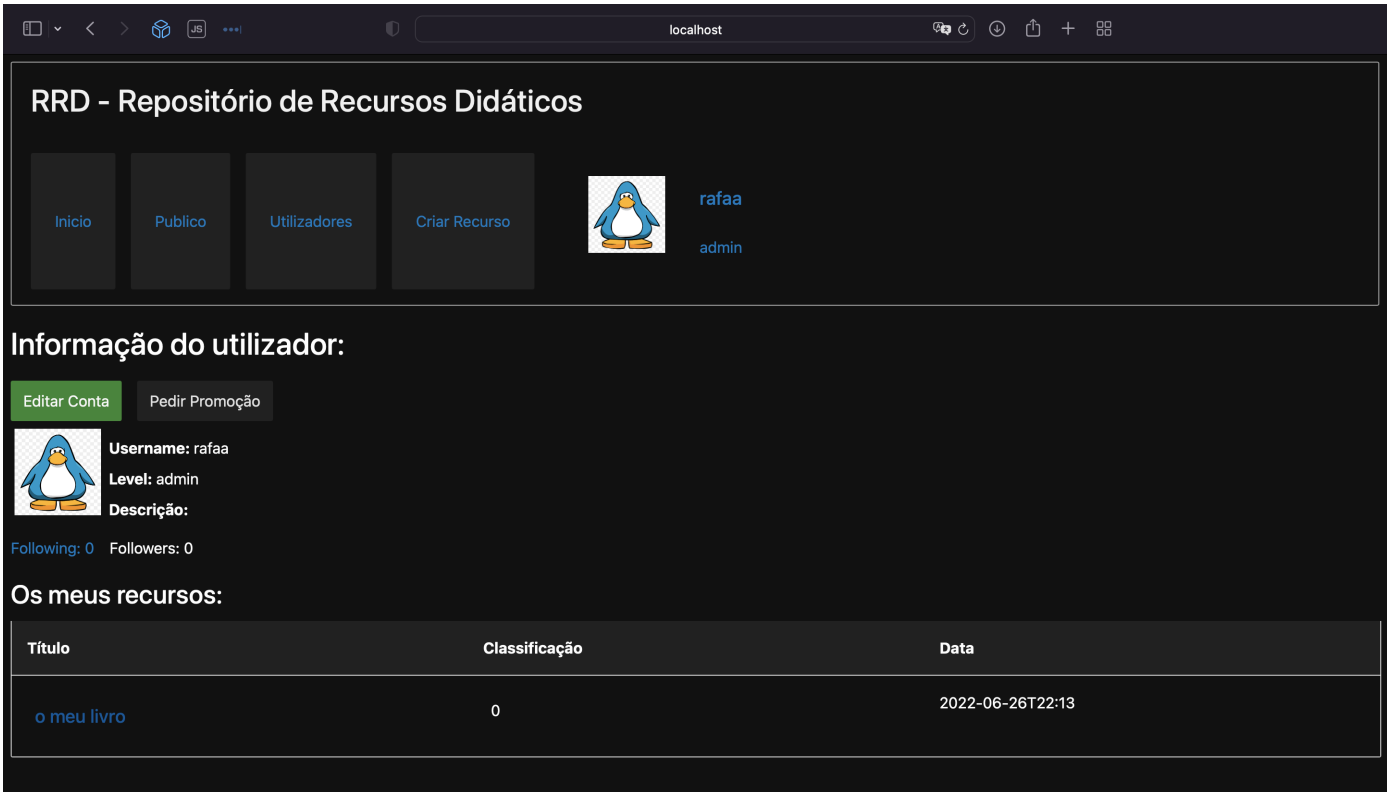

Figura 7: Página do utilizador

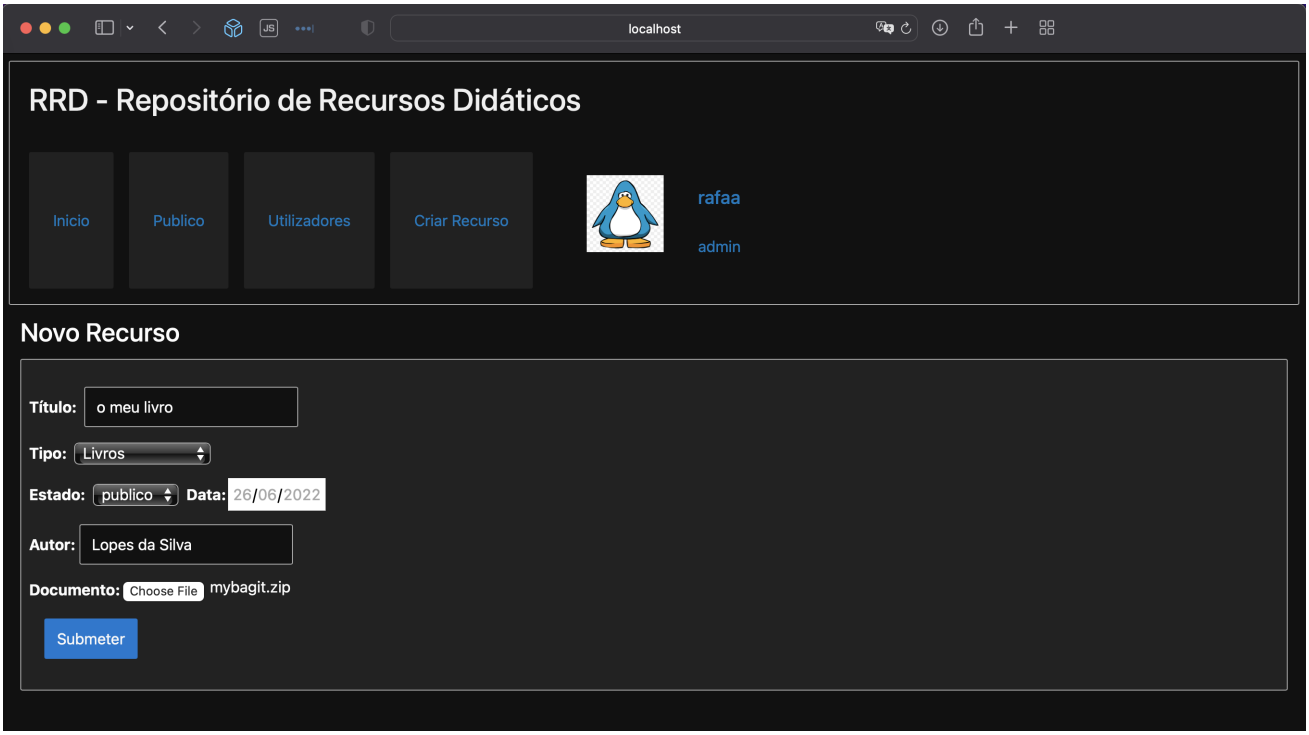

Figura 8: Página para inserção de novo recurso

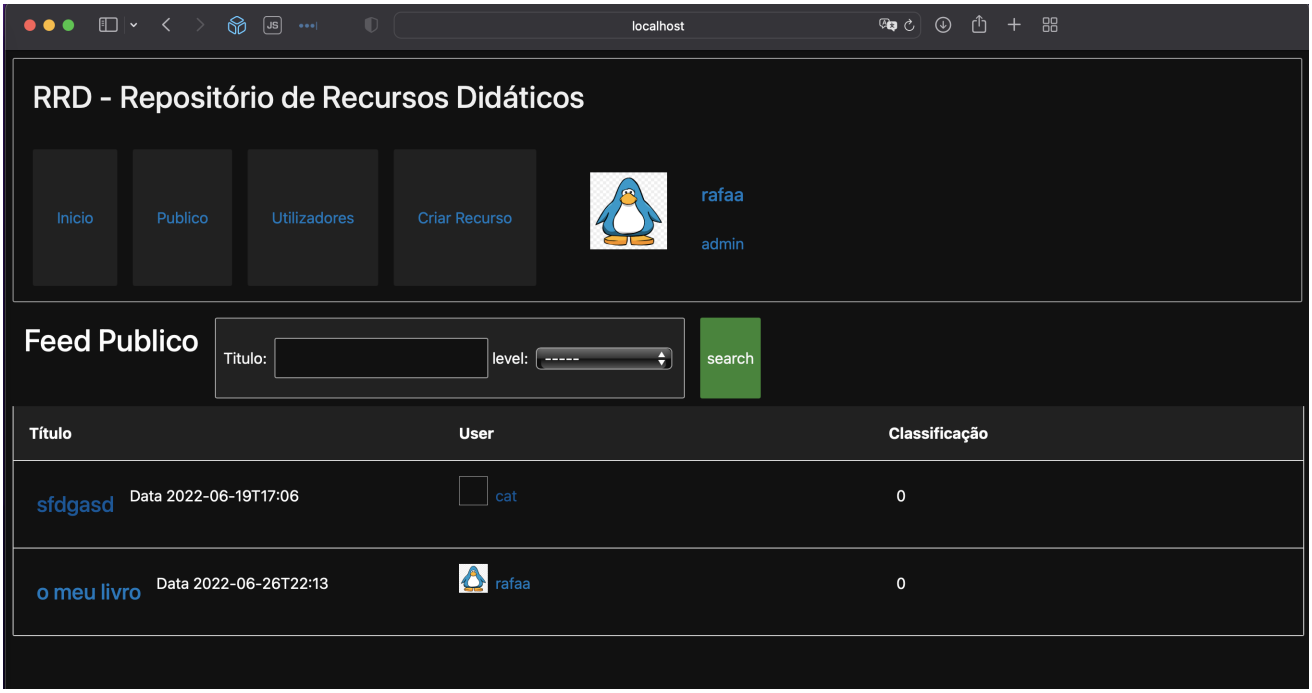

Figura 9: Página com todos os recursos

Figura 10: Página do recurso, parte 1

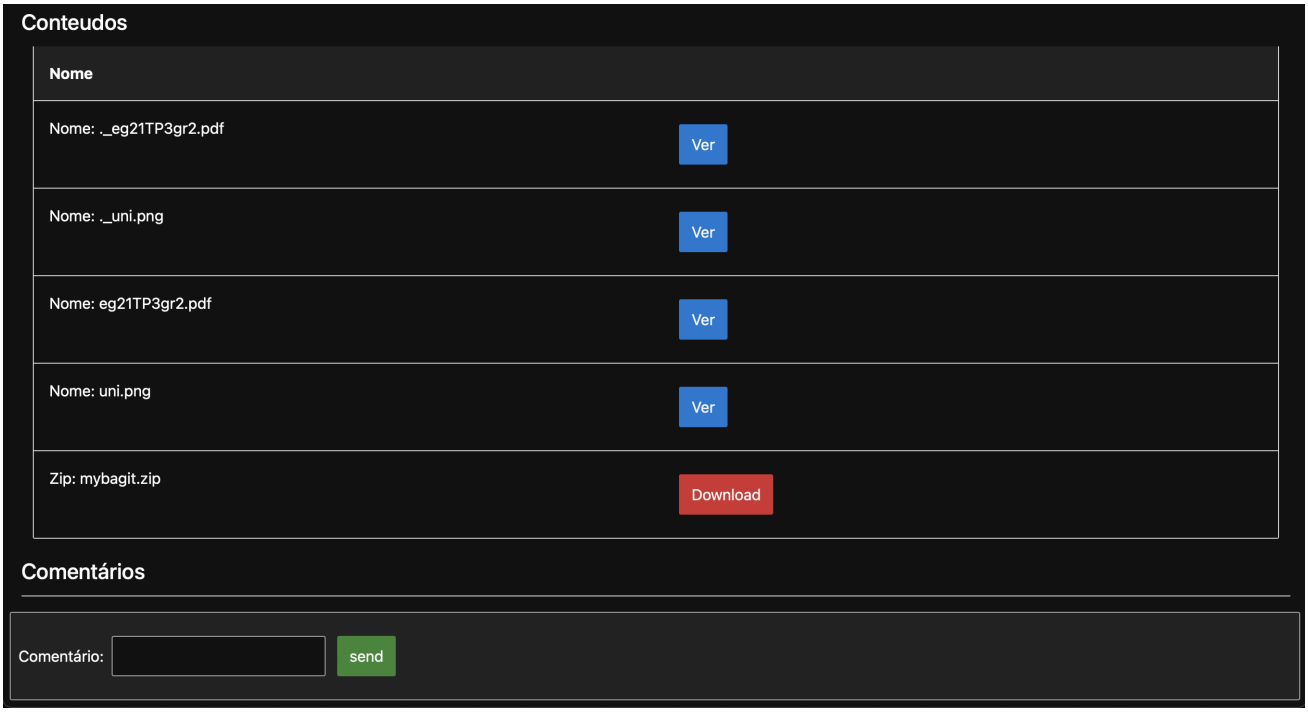

Figura 11: Página do recurso, parte 2

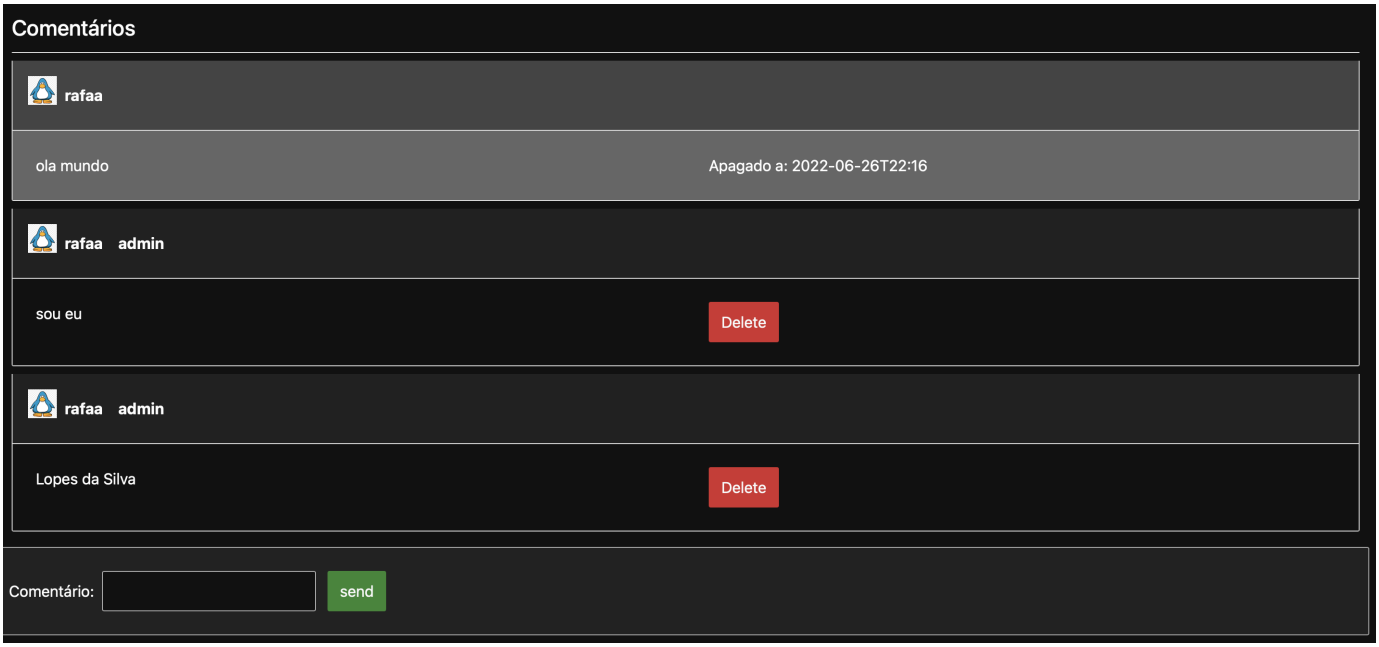

Figura 12: Comentários num recurso

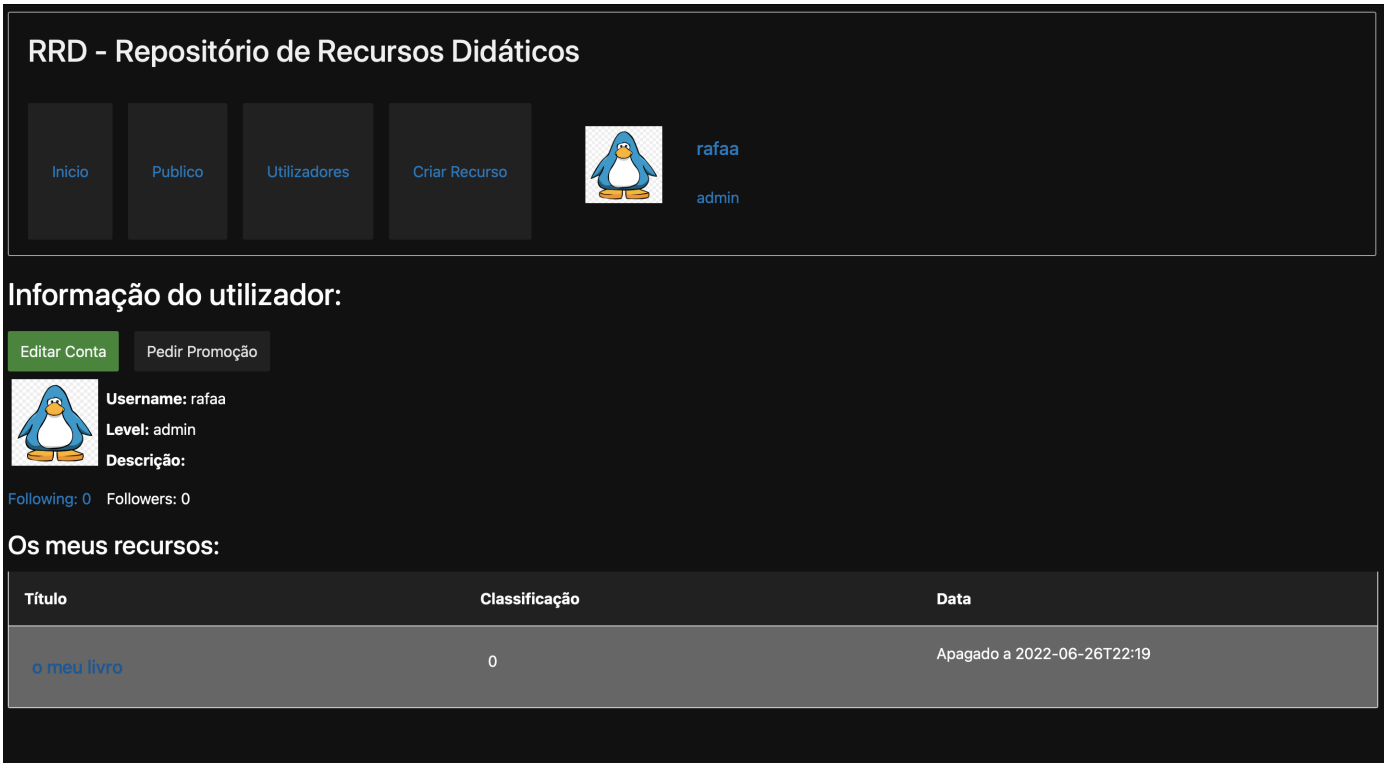

Figura 13: Página do utilizador com recurso apagado

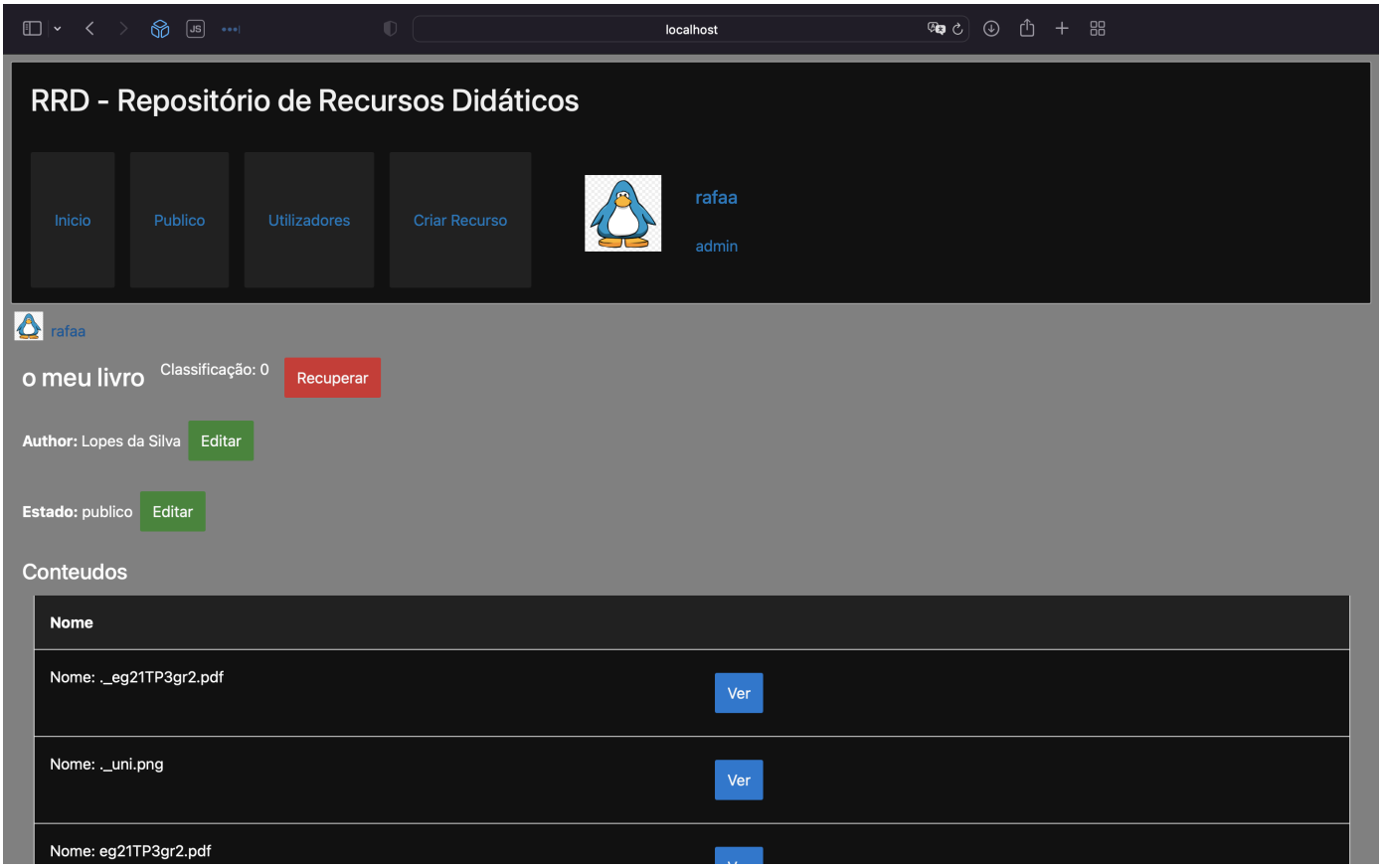

Figura 14: Página de um recurso apagado

5 Conclusão

Terminada a realização deste trabalho prático podemos concluir que se consolidou grande parte da matéria abordada ao longo do semestre na presente unidade curricular.

Algumas das dificuldades encontradas foram a criação das queries para a base de dados, nomeadamente na agregação de *collections* e para a obtenção dos *followers* de um utilizador.

Para além dos requisitos propostos no enunciado, a criação de mais elementos - comentários, seguidores, classificações...- suscitou interesse pela interatividade, aplicabilidade e utilidade que iria criar na aplicação.

Como grupo, não achamos que alcançamos os objetivos que tínhamos estipulado inicialmente, mas que os requisitos propostos foram alcançados. Ficando como objetivo futuramente proposto: alcançar os objetivos inicialmente idealizados.