

Sistema Web para Banda Desenhada

Ana Gil^[a85266], Gonçalo Ferreira^[a84073] e João Abreu^[a84802]

Laboratórios de Engenharia Informática

Universidade do Minho

Orientador: Orlando Belo

Resumo O Cartos é uma plataforma Web para acolhimento e exploração de uma coleção de livros de banda desenhada. No entanto, a plataforma não foi criada de raiz mas sim adaptada de um sistema preexistente chamado "Tommi2". Este sistema era um arquivo de fólios pertencentes à arquidiocese de Braga, que permitia consultar e manipular informação contida nos fólios como também permitia importação de novos documentos para o sistema.

O principal objetivo do grupo foi, não só replicar as funcionalidades do sistema já existente para servir elementos de banda desenhada como também melhorar a experiência de utilização, usando, por exemplo, ficheiros csv para uma rápida e eficiente importação de inúmeros elementos de banda desenhada de uma só vez.

Keywords: Engenharia de Software · Conceção e Desenvolvimento de Software · Plataformas Web · Processamento de Dados · Bases de Dados Orientadas por Grafos · Banda Desenhada.

1 Introdução

Este relatório retrata o trabalho desenvolvido no âmbito da Unidade Curricular de Laboratórios de Engenharia Informática, que teve como objetivo transformar uma plataforma Web existente e adaptá-la para o acolhimento e exploração de uma coleção de livros de banda desenhada. A plataforma já existente designa-se por *Tommi2*, que proveio-o da evolução da plataforma base *Tommi*, e tem como principal propósito o manuseamento da informação que se encontra contida no Livro das Propriedades ou Tombo da Mitra Arquiepiscopal de Braga.

Assim, no contexto do nosso projeto, foi desenvolvida uma plataforma Web, designada por *Cartos*. Através desta plataforma, é possível acolher diversos elementos que caracterizam um livro de banda desenhada, inserindo-os num sistema de bases de dados orientado por grafos. Esta base de dados é explorada pelo sistema desde formulários de pesquisa até *dashboards* de gestão específicos.

Assim, visto que o projeto não foi desenvolvido de raiz, foi necessário fazer uma pré-preparação. Com isto, começamos por nos familiarizar com as tecnologias adotadas no sistema do *Tommi2*, passando por um processo de análise, estudo e pesquisa sobre a sua estrutura e organização. Um dos grandes desafios começou por realizar o processo de migração da base de dados utilizada, passando de uma base de dados documental, *MongoDB* [1], para uma orientada a

grafos, *Neo4J* [2]. De seguida, ao longo do processo de desenvolvimento tivemos de filtrar quais os componentes que não necessitávamos e que não se enquadram no contexto do nosso problema, daqueles que iriam permanecer, fazendo posteriormente as devidas alterações adaptadas ao nosso sistema. Do mesmo modo, foi necessário acrescentar também funcionalidades novas que não estavam presentes na plataforma do *Tommi2*.

2 Aplicação Desenvolvida

2.1 Apresentação

O sistema *Cartos*, tal como referido anteriormente, tem como base o registo e exploração de elementos de banda desenhada. Este pode ser utilizado de três formas: sem efetuar login através da página *Home*, como Leitor e como *Admin*.

Embora permaneça com os mesmos princípios na camada de processamento de informação que o projeto base, a aplicação desenvolvida teve de ser completamente adaptada ao contexto do nosso problema, principalmente na camada de dados.

2.2 Arquitetura

Este projeto, a nível de implementação, está dividido em três camadas:

- Uma base de dados para armazenar toda a informação necessária, sendo esta uma das grande mudanças da nossa plataforma comparativamente à plataforma base.
- Uma camada de processamento de informação para processar e servir dados através de uma API.
- Uma camada de interface, que é responsável por apresentar a informação ao utilizador.

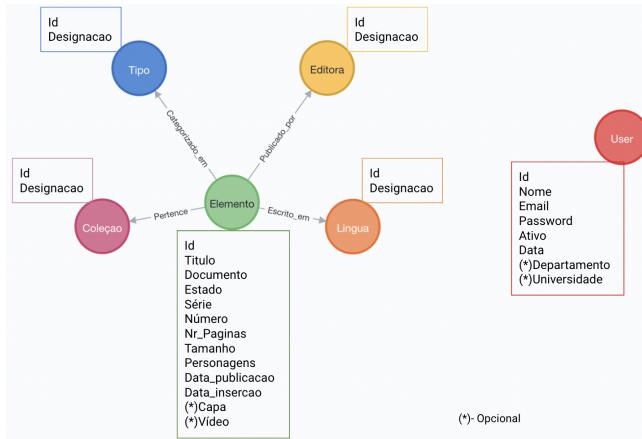
Estas três componentes encontram-se num servidor pertencente à Universidade do Minho, sendo que a API de dados e a interface são servidas à *intranet* da Universidade com Apache2 [3].

Este pode ser acedido em <http://cartos.di.uminho.pt> através das redes da UMinho ou remotamente utilizando a VPN da universidade.

**Figura 1:** Arquitetura do Sistema

2.3 Base de Dados

A base de dados utilizada foi desenvolvida em **Neo4j**, e permite armazenar dados referentes a **utilizadores** e **elementos de banda desenhada**. Estes últimos, para além da informação básica como título, listas de personagens, estado, etc. estão associados a uma **editora**, a um **tipo**, a uma **coleção** e a uma **linguagem**, tratando-se estes últimos de nós independentes na base de dados. São nestas circunstâncias que a escolha de uma base de dados orientada por grafos apresenta uma grande vantagem em comparação com uma base de dados documental, uma vez que nos permite rapidamente devolver, por exemplo, todos os elementos de banda desenhada de uma determinada linguagem. Isto só é possível quando atribuímos um nó independente a estas informações mais relevantes.

**Figura 2:** Arquitetura Base de Dado - Neo4j

2.4 Sistema de Back-End

2.4.1 Configuração

Para o *backend*, utilizamos uma **Rest API** que irá satisfazer os pedidos por parte do *frontend*. Esta arquitetura foi escolhida para facilitar a independência dos componentes sem perder a facilidade de acesso à lógica de negócio. Sendo assim, utilizámos a *framework* **Flask** [4] para a construção de **Rest Controllers** de forma a fazer o mapeamento do pedido e assim obter a resposta adequada. A razão por detrás da escolha desta camada de lógica de negócio deve-se não só pelo projeto anterior ter feito a mesma escolha como também por sugestão do professor orientador. Apesar disso, a ferramenta é conhecida pela sua extensibilidade e rápido processo de desenvolvimento.

2.4.2 Roteamento

O **processamento, inserção e obtenção** dos dados na base de dados através de código *Python* já se encontrava implementado de modo bastante robusto na primeira versão do sistema. Isto permitiu que muito do código presente na versão anterior fosse apenas alterado para que se tornasse compatível com esta nova versão do projeto, onde apenas precisávamos de interagir com a base de dados num novo paradigma. No entanto, algumas das rotas também foram alvo de melhorias na eficiência e legibilidade do código e algumas delas anteriormente desenvolvidas tornaram-se obsoletas, principalmente por não fazerem sentido no contexto do novo problema, e por isso foram eliminadas. Outras rotas foram criadas para melhorar a flexibilidade e funcionalidade do sistema.

Sendo assim, no programa desenvolvido, os *HTTP Requests* são manipulados por **Rest Controllers**. A divisão que achámos mais adequada foi a separação por entidades do sistema. Assim foram criados os seguintes *controllers*:

- **analse** - *Controller* responsável por responder às operações de **procura** por elementos no sistema, juntamente com a consulta do **histórico** de proezas efetuadas.
- **base** - *Controller* que satisfaz os pedidos de autenticação do sistema.
- **elementos** - *Controller* que trata de pedidos de acesso a um **elemento** como também de todas as operações sobre as **dependências** associadas (editoras, tipos, coleções e línguas e também acesso às fotos e vídeos de capa).
- **home** - As operações relacionadas com Home Page do Admin dentro do sistema são mapeadas neste controlador.
- **importacao** - *Controller* que manipula operações de **importação** (criação) e **edição** de elementos de banda desenhada;
- **users** - *Controller* que responde às operações relacionadas com utilizadores, sendo estas a **acesso/criação/remoção** dos mesmos, **atualização** de dados pessoais (currículo e foto de perfil), apresentação do conjunto de pedidos que cada um fez ao sistema e também acesso à lista de utilizadores ativos no sistema.

2.4.3 Autenticação/Segurança

No que diz respeito aos utilizadores, à exceção da autenticação e segurança da API, estes seguem no geral a estrutura implementada anteriormente. Existem dois níveis de acesso para os utilizadores que se autenticam: **administrador** e **leitor**. Um administrador tem controlo total sobre a aplicação, podendo visualizar, alterar, editar ou eliminar toda a informação disponível, tanto a nível de elementos como de outros utilizadores. O leitor tem um acesso mais restrito à aplicação, podendo apenas visualizar a informação disponível dos elementos, realizar pesquisas sobre estes elementos e apenas tendo acesso e controlo sobre o seu próprio perfil de utilização. O sistema, na maioria dos seus métodos, encontra-se protegido por um sistema de autenticação baseado em **JSON Web Token** (JWT) [5], que assegura que apenas utilizadores autorizados podem visualizar ou alterar certa informação do sistema, sendo para tal necessário possuir uma conta e iniciar sessão para poder realizar operações como a inserção de elementos ou a consulta de estatísticas do sistema. Este sistema também se assegura que certos métodos estão apenas disponíveis para administradores, não podendo um leitor, mesmo que autenticado, utilizá-los.

2.4.4 Armazenamento de ficheiros

Tal como foi dito no Capítulo 2.3:

A base de dados utilizada foi desenvolvida em Neo4j, e permite armazenar os dados referentes a utilizadores e elementos de banda desenhada.

Mas a base de dados por si só não armazena **todos** os dados referentes a utilizadores e elementos, pois existem ficheiros que representam cada uma das entidades que não podem ser guardados em base de dados. Este ficheiros são, **fotos de perfil** e **currículos** para o utilizador, e o **pdf** da banda desenhada para os elementos. Sendo assim, estes ficheiros são armazenados no *storage system* do servidor *backend*.

2.4.5 Acesso à Base de Dados

Uma vez que mudamos de base de dados, também tivemos de mudar a maneira como a acedemos. No projeto anterior, o acesso era feito através da biblioteca **PyMongo** [6] e, como pretendíamos usar *Neo4J* para o nosso sistema, tivemos de adaptar todas as ocorrências do seu uso para **py2neo** [7], uma biblioteca que nos permite trabalhar com *Neo4J* a partir do *Python*.

2.5 Sistema de Front-End

Para a construção do *Frontend* da nossa aplicação, fizemos uso da *framework* **Vue.js** [8]. Não só foi usada no projeto anterior como também foi sugerida pelo professor orientador para continuarmos a utilizá-la. Esta escolha deve-se

ao facto desta *framework* ter uma curva de aprendizagem pouco acentuada e ao facto de existir bastante documentação de suporte, bem como a existência de componentes que facilitam certos aspetos da implementação. Algumas destas componentes que utilizamos foram:

- **Vue-Router** - para tratar do roteamento da interface da aplicação.
- **VueCLI** - *npm package* que fornece a capacidade de criar rapidamente um novo projeto ou criar um protótipo instantâneo, através de ferramentas instaladas na linha de comandos.
- **Vuetify** - para ajudar com a cosmética da aplicação.
- **Vue-Store** - para poder armazenar informação acerca do utilizador que se encontra com sessão iniciada.
- **I18n** - para efetuar as traduções de todo o texto da aplicação, o que possibilita a visualização da aplicação em Português, Inglês ou Espanhol.

2.5.1 Components

Uma das filosofias do **Vue.js** é a utilização de componentes reutilizáveis de modo a minimizar a escrita de código repetido e facilitar a manutenção do mesmo. No nosso caso, implementámos diversas componentes, tais como: **Navbars**, **Footers**, **Dialogs**, **Headers**, e outros elementos presentes em diversas páginas.

2.5.2 Views

As **Views** correspondem às diversas páginas a que o utilizador tem acesso, sendo que são também componentes em si. A única diferença entre uma **View** e uma **Componente** prende-se pela questão de uma **View** não poder ser reutilizada noutra *View*, algo que uma Componente consegue. Algumas das principais *Views* por nós implementadas foram: a **Página Inicial**, **perfis dos utilizadores**, páginas de **pesquisa**, página de *login*, formulários de **adição de elementos e utilizadores**, entre outras. No caso dos formulários, recorremos a uma componente bastante útil do *vuetify* chamado de *v-text-field* aliado com regras definidas para cada campo que nos permitiu obter os dados dos formulários com facilidade e validar os mesmos do lado do cliente.

2.5.3 Segurança

No que diz respeito aos utilizadores, foi encontrada a necessidade de guardar localmente a chave de autenticação *JWT*, quando o utilizador realiza o início de sessão na plataforma. Para que esta chave não fosse perdida quando se navega entre diferentes janelas, e de modo a evitar pedidos adicionais à API, foi utilizado o *package vuex-persistedstate*, que permite o armazenamento local da chave enquanto se utiliza a aplicação. Caso esta chave seja alterada indevidamente, ou apagada do sistema, a interface assegura-se de que o utilizador é reencaminhado para a página de iniciar sessão, enquanto que a camada de processamento de

dados rejeita os pedidos que não tenham uma chave válida. Para evitar acessos indevidos, os utilizadores devem terminar sessão da aplicação quando não desejam utilizá-la, visto que esse é o principal método responsável por destruir a chave. Caso não seja feito o término da sessão, ao fim de um certo tempo predefinido, a chave expirará.

2.5.4 API

O *frontend* por si só não apresenta toda a funcionalidade pretendida, sendo que é necessário integrar o *backend*. Para esse efeito, é necessário fazer invocações à nossa **API REST**. Para efetuarmos as invocações à nossa API, e dado que é muito simples a utilização de **JavaScript** puro com **Vue.js**, optámos pela utilização do **Axios** para esse efeito. Outro factor que pesou nesta decisão foi a nossa experiência na utilização do mesmo. Dado que alguns dos pedidos feitos à API requerem autenticação, aquando o início de sessão, guardámos o *token* devolvido nos *headers* do *Axios*, facilitando a utilização do *token*. Aquando ao término de sessão, removemos os valores guardados nos *headers*. Deste modo, somos capazes de integrar nossa API no *frontend* de uma maneira simples e intuitiva.

3 Melhorias

Para além da adaptação feita ao sistema, houve também um esforço por parte do grupo em **melhorar** certos aspetos universais da aplicação independentes do contexto. Alguns desses foram:

3.1 Mudança das rotas login e registo

No sistema antigo, a página de login encontrava-se em `/admin/login` e a de registo em `/admin/registo`. Desta maneira, o utilizador fica com a ideia que apenas *admins* se conseguem autenticar e registar no sistema, e, como não é o caso, optamos por mudar estas 2 rotas para simplesmente `/login` e `/registo`, respetivamente. Para além disso, o sistema preexistente não possuía nenhuma conexão entre as páginas "home", "login" e "registo". Por isso, foram criados botões no cabeçalho de cada uma destas páginas permitindo a navegação entre elas.

3.2 Distinção clara entre Leitor e Admin

Tanto o nosso projeto como o preexistente possuem 2 tipos de utilizadores: **Leitor** e **Admin**. No entanto, havia alguma mistura entre eles que por vezes tornava-se difícil distingui-los. No sistema antigo, tanto um Leitor como um Admin se quisessem aceder à lista dos fólios teriam de navegar para a página que se encontrava em `/admin/folios`. Para além disso, estes utilizadores têm diferentes permissões para interagir com o sistema, ou seja, dentro dessa página

tínhamos de testar que tipo de utilizador era e apresentar só o que cada um podia fazer. No nosso caso, distinguimos as 2 entidades com 2 rotas diferentes: `/leitor/elementos` e `/admin/elementos`, adicionando regras de permissões de acesso a estas páginas (um leitor não consegue aceder a `/admin/elementos` por exemplo). Esta lógica aplica-se para o resto das páginas do sistema, havendo até páginas que só existem no caso do *admin* (por exemplo, `/leitor/import` não existe enquanto que `/admin/import` existe).

3.3 Criação de páginas de erro

Após a implementação das rotas que separavam os 2 tipos de utilizadores do sistema e respetiva segurança associada, surgiu a necessidade de informar ao cliente caso este quisesse aceder a páginas cuja permissão era insuficiente. Sendo assim, criámos páginas de erro para erros **401 Unauthorized** e aproveitamos e criámos também para rotas inexistentes (**404 Not Found**).

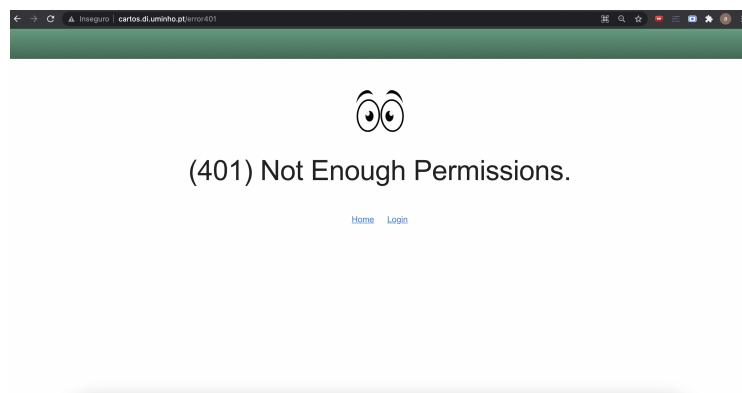


Figura 3: 401 Unauthorized



Figura 4: 404 Not Found

3.4 Importação de múltiplos elementos

Para facilitar e acelerar o processo de inserção de vários elementos para o sistema, desenvolvemos um mecanismo capaz aceitar um ficheiro *csv* com as informações de cada elemento e automaticamente registar na base de dados todos os elementos de uma só vez. No entanto, este processo apenas adiciona uma nova referência na base de dados, não havendo a possibilidade de associar o ficheiro *pdf*, a capa e o vídeo, uma vez que não conseguimos interligá-los diretamente dentro do ficheiro *csv*. Sendo assim, o utilizador terá de ir a cada um dos ficheiros inseridos automaticamente e adicionar, se quiser, um *pdf*, uma capa e um vídeo ao respetivo elemento.

4 Utilização do Sistema

Quando acedemos ao *Cartos* somos inicialmente redirecionados para a página *Home*. Esta permite a um utilizador pesquisar por um ou mais elementos de banda desenhada, quer através da *search bar*, quer através dos filtros selecionados. Uma vez realizada, irá ser apresentada uma lista de resultados consoante a pesquisa feita. É possível, ao clicar sobre um determinado elemento, ver a sua descrição e os respetivos elementos anexados, como o *pdf*, capa e vídeo, sendo os dois últimos elementos opcionais.



Figura 5: Página inicial - *Home*

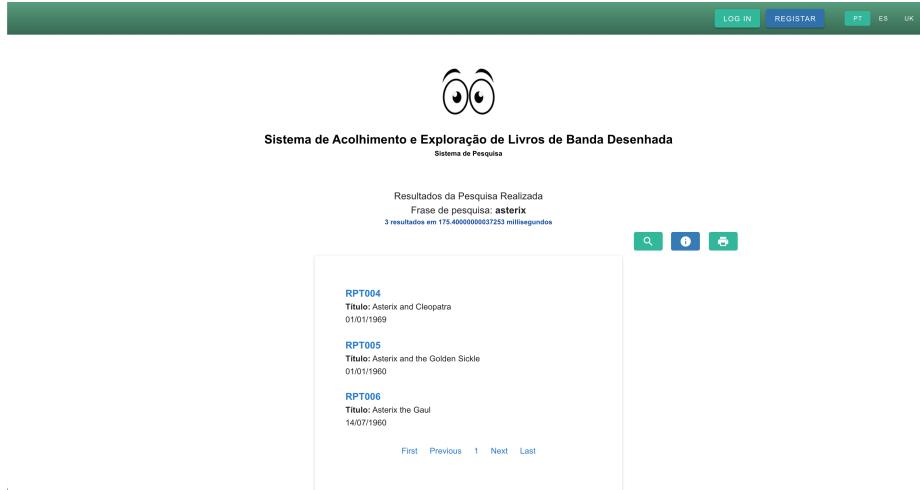


Figura 6: Resultados da pesquisa "asterix".

Como é possível ver pela figura acima (figura 5), no *header* encontram-se três tipos de botões, *LOG IN*, que redireciona o utilizador para a página de inicio de sessão, *REGISTAR*, que permite ao utilizador criar uma conta, e por último temos a funcionalidade de escolher a linguagem do *Website*, estando esta presente ao longo de todas as páginas do sistema.

Tanto a página de autenticação como de registo são bastante simples e funcionam através de formulários, sendo que no caso do registo existem campos obrigatórios e opcionais. Estes estão adaptados para suportar falhas, como por exemplo, a invalidade das credenciais de acesso, no caso do login, e a obrigatoriedade de alguns campos assim como a não repetição de *usernames*, no caso do registo. Para estes casos é dado um feedback ao utilizador a informar a devida falha.

Figura 7: Página de autenticação - *Log In*

Figura 8: Página de registo

Tal como referido anteriormente, existem dois tipos de utilizadores, o *Leitor* e o *Admin*. Enquanto que o *Admin* possui controlo total sobre o sistema, o *Leitor* apenas consegue realizar uma parte das operações deste. Por este motivo, iremos continuar a apresentação desta secção tendo por base o tipo de conta *Admin*.

Após o início de sessão efetuado com sucesso, é apresentada a página inicial do utilizador, na qual é apresentado um cabeçalho, menu de navegação lateral e a secção de conteúdo.

O cabeçalho passa conter um botão de menu que permite abrir/fechar o menu lateral, o Logo da aplicação, o Título, o utilizador em sessão com a sua foto e respetiva informação e também os botões de idiomas. A navegação lateral lista os diversos conteúdos que o sistema disponibiliza, que ao serem selecionados são renderizados na secção de conteúdo.



Figura 9: Menu de Navegação Lateral

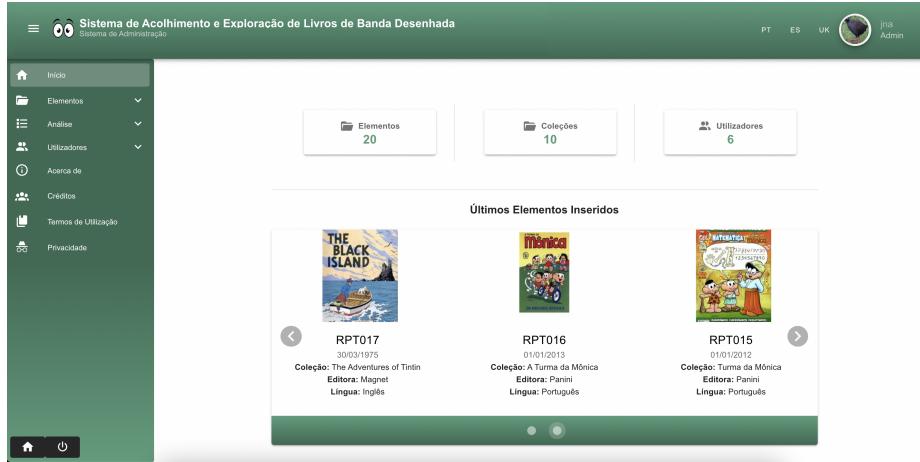


Figura 10: Página de inicial.

O ícone do utilizador ao ser premido permite ainda sair de sessão e também visualizar a sua informação pessoal como também atualiza-la.

A página inicial por defeito mostra alguns dados estatísticos entre eles o número de elementos, coleções e utilizadores, também gráficos que representam a relação entre Elementos por coleção e editora, apresenta ainda um carrossel dos últimos elementos adicionados ao sistema.

Os conteúdos apresentados no menu de navegação lateral permitem realizar diversas operações no sistema, as quais passam a ser listadas:

- **Início¹** ([figura 10](#)): a página inicial acima apresenta uma visão geral do sistema de banda desenhada.
- **Elementos**: nesta aba é possível observar sub-funcionalidades que pertencem aos Elementos.
 - **Gestão de Elementos¹**. ([figura 17](#)): permite consultar, editar, apagar e também inserir novos elementos de banda desenhada. No caso do utilizador autenticado ser um Leitor, nesta página este apenas consegue consultar elementos.
 - **Importação²** ([figura 18](#)): é responsável por importar múltiplos elementos através de um ficheiro *csv* a detalhar propriedades respetivas a cada um.
- **Análise**: nesta aba é possível observar funcionalidades que pertencem à Análise do Sistema.
 - **Pesquisa¹** ([figura 19](#)): possibilita efetuar pesquisas no sistema para consultar bandas desenhadas de acordo com algumas escolhas para filtragem do resultado. Esta página segue a mesma lógica que a página

¹ Acessível pelo Admin e Leitor

² Acessível apenas pelo Admin

Home, apresentada anteriormente e presente na figura 5. A existência desta página permite efetuar pesquisas sem ter de sair da área autenticada, juntamente com a associação entre o utilizador e a pesquisa feita.

- **Pesquisas Realizadas²** (figura 20): permite rever pesquisas anteriormente feitas, sabendo que utilizador a fez (no caso da pesquisa ter sido feita dentro da área autenticada, caso contrário, é atribuído como anónimo), a data quando foi realizada e os parâmetros da mesma.
- **Utilizadores**: nesta aba é possível observar funcionalidades que pertencem aos Utilizadores.
 - **Gestão de Utilizadores²** (figura 21): consulta das informações sobre utilizadores presentes no sistema, visualização dos seus ficheiros associados como por exemplo currículo e foto, possibilidade de alterar dados pessoais e remoção do sistema dos mesmos.
 - **Utilizadores Ativos²** (figura 22): permite visualizar os utilizadores atualmente ativos no sistema, juntamente com a última data de acesso.
 - **Historial de Acesso²** (figura 23): analisar o histórico de navegação dos diversos utilizadores no sistema, sabendo que *HTTP Requests* cada utilizador fez e quando.
- **Acerca de¹** (figura 24): *pop-up* com um texto com o intuito da aplicação.
- **Créditos¹** (figura 25): *pop-up* com as pessoas envolvidas no desenvolvimento deste projeto.
- **Termos de Utilização¹** (figura 26): *pop-up* contendo a propriedade do *software* em uso como também os seus termos de utilização.
- **Privacidade¹** (figura 27): *pop-up* que explica a política de privacidade e processamento de dados por parte da aplicação.

Visto que os elementos são dados cruciais para utilização do sistema, é descrito o procedimento para inserir um elemento no site.

Primeiro, é necessário abrir o menu lateral e a aceder à área de gestão de elementos. Em seguida, é apresentado uma tabela com os elementos existentes no sistema, e também nota-se que existe um botão característico para a adição de um novo elemento de banda desenhada no canto superior direito (como se pode ver na figura 17). Ao pressionar esse botão é renderizada a pagina de inserção de um elemento que é possível observar na imagem abaixo.

Figura 11: Importar Elemento de Base de dados.

O preenchimento dos campos segue um conjunto de regras, tais como:

- Preenchimento obrigatório de todos os campos, excepto capa e vídeo;
- O identificador necessita de começar por "RPT", seguido de 3 dígitos obrigatoriamente;
- A data deve ter o formato: DD/MM/AAAA.
- Os valores inseridos não devem ultrapassar o limite de caracteres estabelecido.

De forma a auxiliar o processo de inserção, é disponibilizado o botão "Importar". Basicamente, este importa os parâmetros base de um dado elemento já existente no sistema e preenche-os nos devidos campos do formulário atual.

5 Conclusões e Trabalho Futuro

O desafio de adaptar a plataforma *web Tommi2* teve como primeira abordagem a análise cuidada do sistema base. A compreensão e estudo progressivo do *software* surge naturalmente com objetivo da adaptação. A arquitetura adotada segue a norma de três camadas que facilita o desenvolvimento e qualidade do produto, isto deve-se à especificidade dos serviços para o *front-end*, *back-end* e base de dados.

A alteração com o novo conceito é a persistência de dados orientada a grafos, que proporciona uma melhor utilização tendo em conta as relações entre a informação do sistema. Com uma rápida aprendizagem e uso da biblioteca *py2neo* foi possível ultrapassar este obstáculo, garantindo assim um fácil manuseamento dos dados.

Em segundo lugar, após a familiarização com as ferramentas *Flask*, *JWT*, *Apache2*, passou-se à reestruturação do *back-end* com objetivo de alterar os

acessos a dados persistidos, como também integração de funcionalidades acrescidas. Posteriormente, a interface gráfica sofreu alterações devido ao intuito da aplicação. As tecnologias *Vue.js* e *Vuetify* aceleraram o desenvolvimento da plataforma *web* devido à funcionalidade de separação por componentes e ao seu fácil uso.

Para sintetizar, o desenvolvimento da plataforma com o uso das diversas tecnologias demonstrou inicialmente alguns obstáculos. No entanto, estes são facilmente ultrapassados devido aos vastos recursos de documentação, como também serem tecnologias muito utilizadas com comunidades grandes. Isto, possibilitou a plataforma reter as vantagens do uso das tecnologias como também a longevidade e manutenção, de modo a aumentar a viabilidade da solução.

Com o trabalho futuro em vista, existem aspectos que mesmo consolidados poderiam ter sido executados com uma perspectiva mais robusta. Por exemplo, o sistema de pesquisas auxiliar-se de métodos mais complexos de procura como também o apoio de um sistema de recomendação apropriado. Outro exemplo seria na importação automática de múltiplos elementos através do ficheiro *csv* conseguirmos associar o ficheiro *pdf*, o vídeo e a capa colocando o *path* para cada um e depois o respetivo upload para o *storage system*.

O resultado do trabalho vai de encontro aos requisitos solicitados, mesmo com espaço para melhoria. Conclui-se que a aplicação teve uma adaptação positiva ao sistema base, aprimorando os detalhes necessários para produzir um sistema de gestão de base de dados completamente funcional e estável.

6 Anexos

6.1 Documentação API Flask

The screenshot shows the Swagger UI interface for an API. At the top, there's a header with the Swagger logo, the URL '/apispec_1.json', and a 'Explore' button. Below the header, it says 'A swagger API 0.3.1' and 'powered by Flask-Spec'. There are links for 'apispec_1.json' and 'Terms of service'. The main content area is titled 'default' and lists several GET requests:

- /analise/pesquisa - Efetuar uma nova pesquisa.
- /analise/pesquisas - Consultar pesquisas armazenadas efetuadas.
- /elementos/apagar/{elemento} - Apagar elemento.
- /elementos/colecoes - Get Coleções.
- /elementos/editoras - Get Editores.
- /elementos/elementos - Get Elementos no Sistema.
- /elementos/linguas - Get Línguas.
- /elementos/tipos - Get Tipos.
- /elementos/ver/{elemento}/ficheiro - Ver ficheiro do elemento.
- /elementos/ver/{elemento}/foto - Ver foto do elemento.
- /elementos/ver/{elemento}/video - Ver vídeo do elemento.

Figura 12: Página inicial da documentação.

This screenshot shows the detailed documentation for a POST request to the '/login' endpoint. At the top, it says 'POST /login Iniciar Sessão.' and has a 'Try it out' button. Below that, there's a 'Parameters' section with two fields: 'id' (required, string) and 'password' (required, string). Under the 'Responses' section, there are two entries:

- Code 200: Successo a efetuar login.
Example Value : Model
JSON example:


```
{
        "token": "string",
        "user": {}
      }
    
```
- Code 500: Insucesso a efetuar login.
Example Value : Model
JSON example:


```
{
        "error": "string"
      }
    
```

Figura 13: Documentação do POST da rota '/login'.

POST /login Iniciar Sessão.

Parameters

Name	Description
<code>id * required</code>	obeto
<code>password * required</code>	Obeto

Responses

Curl

```
curl -X POST "http://ssh.tomil2.di.uninho.pt:5002/login" -H "Accept: application/json" -H "Content-Type: application/x-www-form-urlencoded" -d "id=obeto&password=obeto"
```

Request URL

<http://ssh.tomil2.di.uninho.pt:5002/login>

Server response

Code Details

200

Response body

```
{"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ejZrDNi161jYmNsb3VlLhdG1MTyNwemjEzh1M01MmZfQ.Gq1d70n2AC1x5s9U0ZcrlxYbxE7mpxtY723i4g", "user": {"obs": "Obe", "name": "Obeto", "email": "obeto@gmail.com", "departamento": "Informática", "id": "obeto", "id_c": "obeto", "password": "Obeto", "universidade": "Minho", "stamp": "2021-06-18 22:33:10.175144"}, "server": {"ip": "192.168.1.11", "port": "5002", "version": "2.0.0", "language": "Python", "os": "Ubuntu 22.04 LTS", "ram_gb": 8.0}, "date": "Tue, 22 Jun 2021, 17:10:36 GMT", "server_stamping": "0.132", "version": "2.0.0", "university": "Universidade de Minho", "stamp": "2021-06-22 16:59:14.659031", "departamento": "Informática", "name": "Jma", "id": "jma", "email": "jmag@gmail.com"}}, {"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ejZrDNi161jYmNsb3VlLhdG1MTyNwemjEzh1M01MmZfQ.Gq1d70n2AC1x5s9U0ZcrlxYbxE7mpxtY723i4g", "user": {"obs": "Obe", "name": "Obeto", "email": "obeto@gmail.com", "departamento": "Informática", "id": "obeto", "id_c": "obeto", "password": "Obeto", "universidade": "Minho", "stamp": "2021-06-18 22:33:10.175144"}, "server": {"ip": "192.168.1.11", "port": "5002", "version": "2.0.0", "language": "Python", "os": "Ubuntu 22.04 LTS", "ram_gb": 8.0}, "date": "Tue, 22 Jun 2021, 17:10:36 GMT", "server_stamping": "0.132", "version": "2.0.0", "university": "Universidade de Minho", "stamp": "2021-06-22 16:59:14.659031", "departamento": "Informática", "name": "Jma", "id": "jma", "email": "jmag@gmail.com"}}
```

Response headers

```
access-control-allow-origin: http://ssh.tomil2.di.uninho.pt:5002
content-type: text/html; charset=utf-8
date: Tue, 22 Jun 2021, 17:10:36 GMT
server: stamping/0.132
vary: Origin
```

Responses

Figura 14: Experimentar o POST da rota '/login'.

GET /elementos/{elemento} - Get Elemento por ID.

Parameters

Name	Description
<code>elemento * required</code>	elemento

Responses

Code Description

200 Lista com o Elemento correspondente.

Example Value

```
[{"id": "elemento", "name": "elemento", "obs": "Elemento de teste", "stamp": "2021-06-18 22:33:10.175144", "university": "Universidade de Minho", "departamento": "Informática", "email": "elemento@gmail.com", "password": "Elemento", "series": [{"id": "series", "text": "Série 1", "type": "string", "stamp": "2021-06-18 22:33:10.175144"}, {"id": "series", "text": "Série 2", "type": "string", "stamp": "2021-06-18 22:33:10.175144"}, {"id": "series", "text": "Série 3", "type": "string", "stamp": "2021-06-18 22:33:10.175144"}]}]
```

Figura 15: Documentação do GET da rota '/elemento/<elemento>'.

```
+ curl -X GET "http://ssh.tommi2.di.uminho.pt:5002/elementos/RPT012" -H "accept: application/json"
[{"id": "RPT012", "titulo": "Tintin in Tibet", "capa": "", "estado": "Bom", "numero": "5", "nr_paginas": "62", "texto": "RPT012.pdf", "observacoes": "", "tamanho": "A4", "personagens": "Tintin, Captain Haddock, Milu", "serie": "Tintin", "data_publicacao": "01/01/1975", "colecao": "The Adventures of Tintin", "editora": "Magnet", "lingua": "Ingl\u00e9s", "tipo": "Aventura"}]
```

Figura 16: Curl gerado do GET da rota '/elemento/< elemento >'.

6.2 Páginas de Administração

Identificador	Coleção	Editora	Data	Língua	Opcões
RPT001	A Lucky Luke Adventure	Paw Prints	2008/01/01	Inglês	
RPT002	A Lucky Luke Adventure	Dupuis	1988/01/01	Francês	
RPT003	A Lucky Luke Adventure	Dupuis	2005/01/01	Francês	
RPT004	Astérix	Hachette Dargaud	1969/01/01	Inglês	
RPT005	Astérix	Hachette Dargaud	1969/01/01	Inglês	
RPT006	Astérix	Hachette Dargaud	1990/07/14	Inglês	
RPT007	Batman	DC	2011/01/01	Inglês	
RPT008	Spiderman	Marvel	1962/01/01	Inglês	
RPT009	DC Marvel Comics	DC Marvel	1995/01/01	Inglês	
RPT010	Donald Duck	Boom Kids	2010/12/01	Inglês	
RPT011	Donald Duck	Boom Kids	2012/03/01	Inglês	
RPT012	The Adventures of Tintin	Magnet	1973/01/01	Inglês	
RPT013	The Adventures of Tintin	Distribuidora Record	1970/01/01	Português	
RPT014	The Adventures of Tintin	Magnet	1978/01/01	Inglês	

Figura 17: Página de gestão de Elementos.

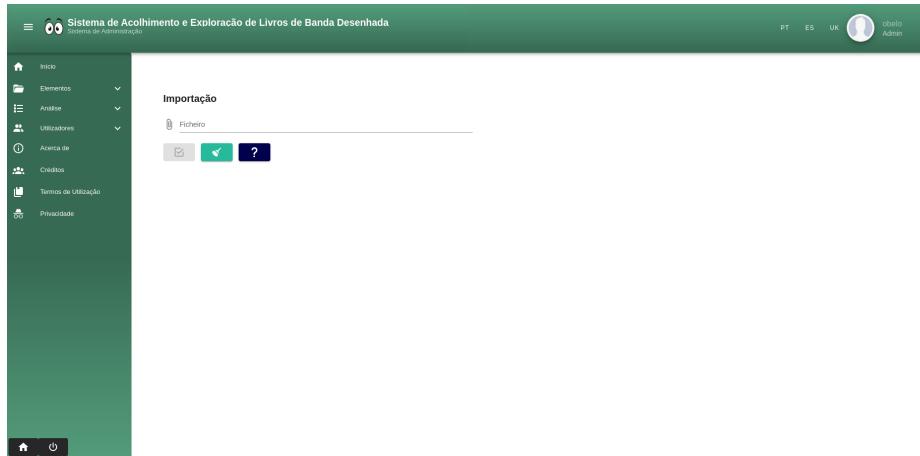
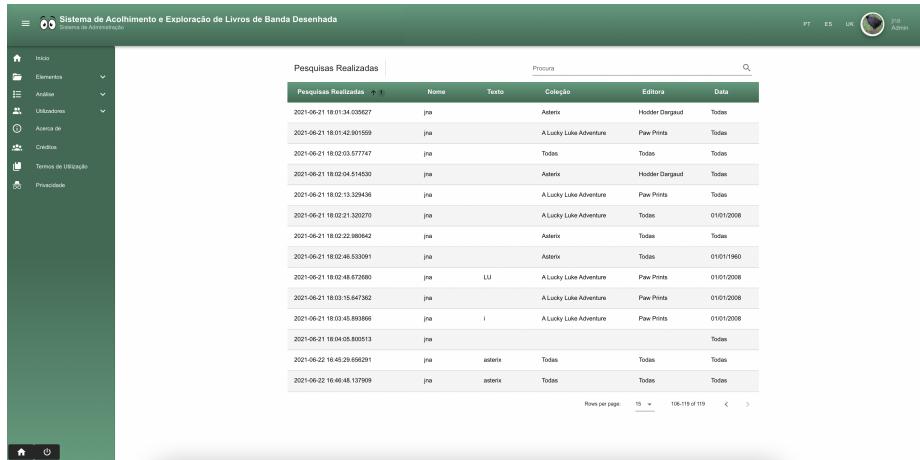


Figura 18: Página de importação de Elementos.

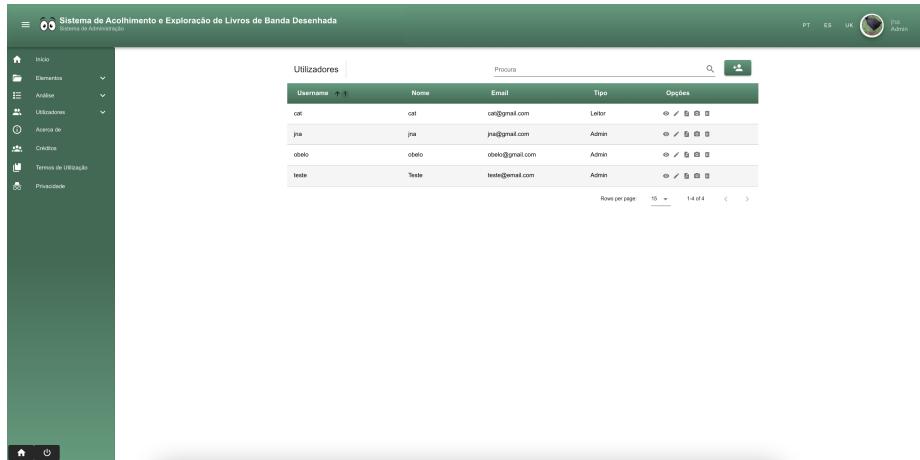


Figura 19: Página de análise de pesquisas.



The screenshot shows a table titled 'Pesquisas Realizadas' (Searches Made) with the following data:

Pesquisa Realizada	Nome	Texto	Coleção	Editora	Data
2021-06-21 18:01:34.035527	jra	Asterix	Hodder Dargaud	Todas	
2021-06-21 18:01:42.901569	jra	A Lucky Luke Adventure	Paw Prints	Todas	
2021-06-21 18:02:03.577747	jra		Todas	Todas	
2021-06-21 18:02:04.514530	jra	Asterix	Hodder Dargaud	Todas	
2021-06-21 18:02:13.329438	jra	A Lucky Luke Adventure	Paw Prints	Todas	
2021-06-21 18:02:21.320270	jra	A Lucky Luke Adventure	Todas	01/01/2008	
2021-06-21 18:02:22.980642	jra	Asterix	Todas	Todas	
2021-06-21 18:02:46.533091	jra	Asterix	Todas	01/01/1960	
2021-06-21 18:02:48.672680	jra	LU	A Lucky Luke Adventure	Paw Prints	01/01/2008
2021-06-21 18:03:15.647362	jra	A Lucky Luke Adventure	Paw Prints	01/01/2008	
2021-06-21 18:03:45.893956	jra	i	A Lucky Luke Adventure	Paw Prints	01/01/2008
2021-06-21 18:04:05.800513	jra			Todas	
2021-06-21 16:45:29.656291	jra	asterix	Todas	Todas	Todas
2021-06-22 16:46:48.137909	jra	asterix	Todas	Todas	Todas

Figura 20: Página de análise de pesquisas realizadas.


The screenshot shows a table titled 'Utilizadores' (Users) with the following data:

Username	Nome	Email	Tipo	Opcões
ot	ot	ot@gmail.com	Lector	<input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
jra	jra	jra@gmail.com	Admin	<input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
obeto	obeto	obeto@gmail.com	Admin	<input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
teste	teste	teste@gmail.com	Admin	<input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

Figura 21: Página de gestão de utilizadores.

Utilizadores Ativos	
Nome	Data de acesso
obelo	2021-06-23 15:55:12.225130
jna	2021-06-23 15:57:56.624281

Figura 22: Página de gestão de utilizadores ativos.

Data de acesso	Nome	Pedido
2021-06-22 18:53:07.950020	obelo	GET http://carlos.dl.uminho.pt/users/history?name=obelo
2021-06-22 18:52:38.569553	obelo	GET http://carlos.dl.uminho.pt/users/active?name=obelo
2021-06-22 18:52:24.237484	obelo	GET http://carlos.dl.uminho.pt/analise/pesquisas
2021-06-22 18:20:54.519486	obelo	GET http://carlos.dl.uminho.pt/users/foto/obelo
2021-06-22 18:59:14.659031	jna	GET http://carlos.dl.uminho.pt/users/history?seed=162437754409
2021-06-22 16:45:19.744887	jna	GET http://carlos.dl.uminho.pt/users/foto/jna
2021-06-21 18:52:44.639306	obelo	GET http://carlos.dl.uminho.pt/users/foto/cat
2021-06-21 18:26:38.826766	jna	GET http://carlos.dl.uminho.pt/analise/pesquisas
2021-06-21 18:09:31.327496	jna	GET http://carlos.dl.uminho.pt/users/history?name=jna
2021-06-21 17:52:24.808099	jna	GET http://carlos.dl.uminho.pt/users/circulo/cat?seed=1624294344671
2021-06-21 17:52:18.030013	jna	GET http://carlos.dl.uminho.pt/users/circulo/last?seed=1624294337895
2021-06-21 17:52:14.434523	jna	GET http://carlos.dl.uminho.pt/users/foto/ads?seed=1624294334109
2021-06-21 17:52:13.375347	jna	GET http://carlos.dl.uminho.pt/users/foto/ads?seed=1624294333064

Figura 23: Página de gestão de pedidos dos utilizadores.

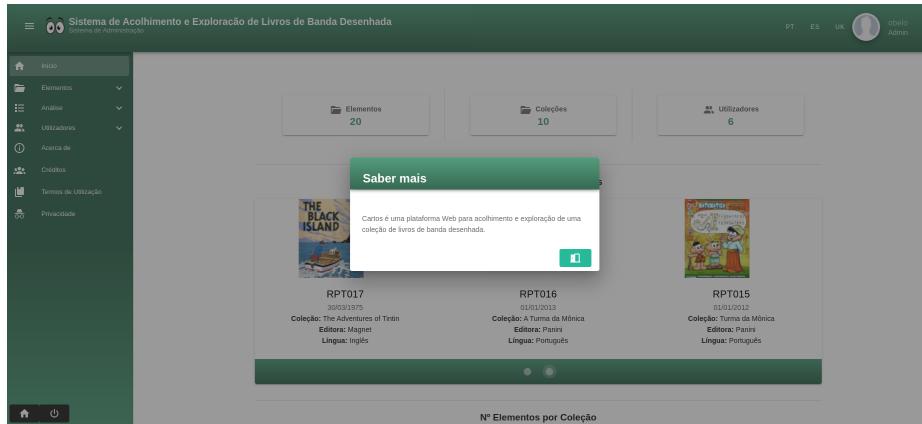


Figura 24: Pop-up da aba 'acerda de'.

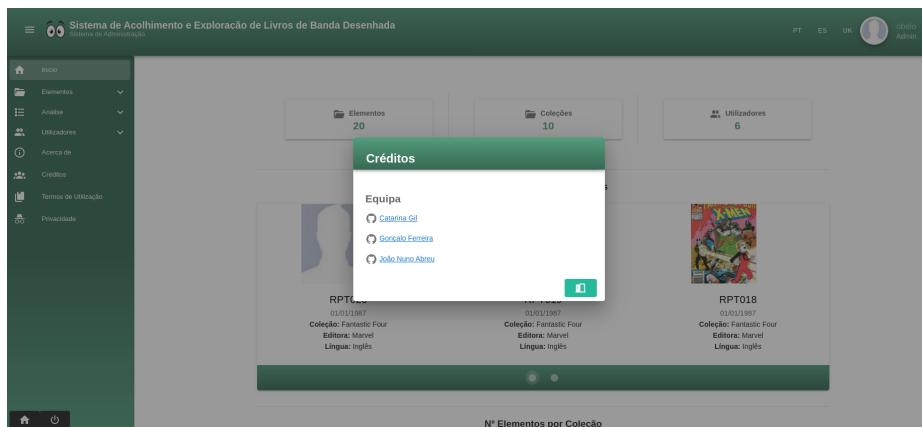


Figura 25: Pop-up da aba 'créditos'.

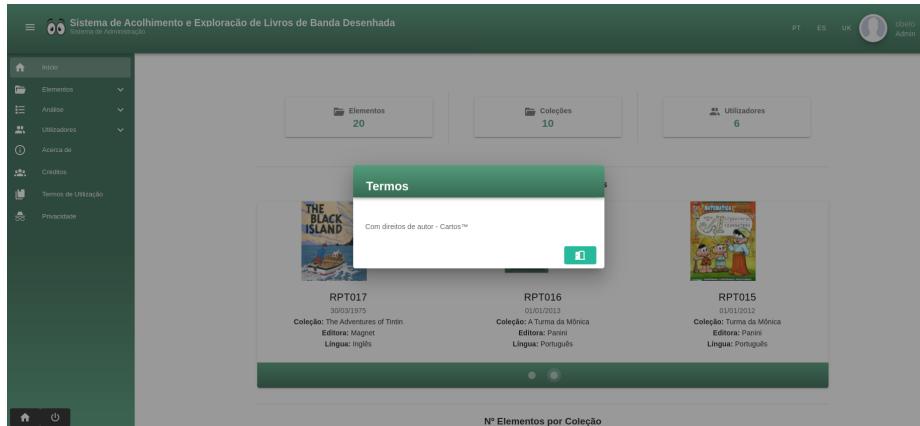


Figura 26: Pop-up da aba 'termos de utilização'.

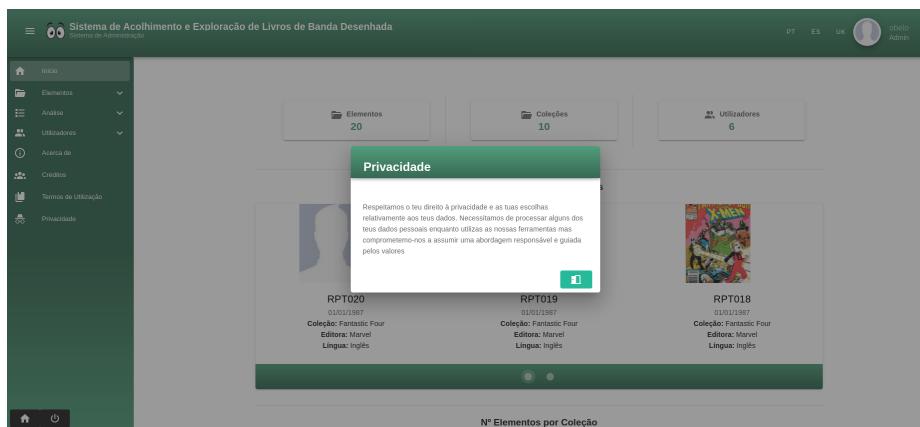


Figura 27: Pop-up da aba 'privacidade'.

Referências

1. MongoDB, <https://www.mongodb.com/>
2. Neo4j, <https://neo4j.com/>
3. The Apache HTTP Server Project, <https://httpd.apache.org/>
4. Flask, <https://flask.palletsprojects.com/en/2.0.x/>
5. JSON Web Tokens, <https://jwt.io/>
6. PyMongo, <https://pymongo.readthedocs.io/en/stable/>
7. The Py2neo Handbook, <https://py2neo.org/2021.1/>
8. Vue.js, <https://vuejs.org/>