Representação e Processamento de Conhecimento na Web
(1º ano do MEI)

Repositório de Recursos Didáticos (RRD)

Relatório de Desenvolvimento

João Pereira	Luís Vieira	Pedro Barbosa
PG47325	PG47430	PG47577

28 de junho de 2022

Resumo

O presente trabalho prático, realizado no âmbito da unidade curricular de Representação e Processamento de Conhecimento na Web inserida no perfil de Engenharia de Linguagens, visa o desenvolvimento de uma aplicação web que implemente e dê suporte a um repositório de recursos didáticos. Este irá começar com uma introdução sobre o projeto prático. De seguida, irá ser abordada e discutida a sua arquitetura aplicacional. Posteriormente, serão explicados os diferentes tipos de utilizadores bem como as suas permissões na aplicação e a implementação dos diferentes tipos de servidores. Finalmente, serão apresentados resultados e será feita uma conclusão sobre os mesmos.

Conteúdo

1 Introdução **2**

2 Arquitetura aplicacional **4**

3 Utilizadores e permissões **5**
3.1 Administrador 5
3.2 Consumidor 5
3.3 Produtor 6

4 Implementação dos servidores **7**
4.1 API-Server 7
4.1.1 Modelos 7
4.1.2 Controladores 9
4.1.3 Roteadores 9
4.1.4 Proteção das rotas 10
4.2 App-Server 10
4.2.1 Roteadores 11
4.2.2 Pacotes de Informação 13
4.2.3 File Storage 13
4.3 Auth-Server 14
4.3.1 Modelos 14
4.3.2 Controladores 14
4.3.3 Roteadores 14
4.3.4 Estratégia de autenticação 16
4.4 Log-Server 16

5 Resultados **17**

6 Conclusão **24**

Capítulo 1

Introdução

O desenvolvimento do tema proposto insere-se na continuação do estudo feito durante as aulas práticas da unidade curricular. Assim, foi posto em prática o conhecimento adquirido sobre alguns módulos e ferramentas aprendidas ao longo do semestre. Desta forma, no que concerne ao trabalho prático, o seu principal objetivo consiste em desenvolver uma aplicação web de um repositório de recursos didáticos (RRD).

Posto isto, a aplicação Web concebida compreende as seguintes principais funcionalidades:

- *Upload* e *Download* de diferentes tipos de recursos.
- Consulta de recursos de difentes tipos.
- Consulta de notícias.
- Comentar e colocar *likes* em recursos.
- Registo de *logs* e a sua respetiva consulta e processamento.

Assim sendo, e tendo em consideração todos os tópicos acima abordados, este relatório visa ajudar a compreender e explicar todos os raciocínios e tomadas de decisão feitas de forma a conseguir realizar todas as tarefas pedidas neste projeto e, assim, atingir o resultado final desejado.

Estrutura do Relatório

Este relatório inicia-se no capítulo 1 onde é feita uma contextualização e enquadramento do projeto prático e uma apresentação das funcionalidades que devem existir aquando da conclusão do mesmo.

Segue-se do capítulo 2 onde é feita uma abordagem da arquitetura aplicacional concebida para o desenvolvimento da aplicação bem como uma pequena explicação de cada um dos servidores existentes.

Posteriormente, no capítulo 3 será explicada a concepção idealizada para a cada um dos tipos de utilizadores existentes bem como as funcionalidades a que cada um destes tem acesso consoante as suas permissões.

Em seguida, no capítulo 4 serão debatidas as principais tomadas de decisão efetuadas para a conceção dos diferentes tipos de servidores desenvolvidos.

Subsequentemente, no capítulo 5 serão apresentados e discutidos os resultados obtidos aquando a finalização da realização do projeto prático.

Finalmente o relatório termina com o capítulo 6 onde é feita uma síntese do documento e uma análise crítica generalizada do trabalho e dos resultados obtidos.

Capítulo 2

Arquitetura aplicacional

A solução arquitetural concebida para o projeto prático baseia-se em micro-serviços. Assim sendo, existem três tipos de servidores diferentes:

- O **api server** é o servidor responsável por responder com informações relativas aos recursos armazenados;
- O **application server** foi desenvolvido para permitir ao utilizador interagir com a aplicação e ter acesso à interface gráfica da mesma, desta forma este servidor responde com páginas web de acordo com o que o utilizador pretender consultar;
- Por fim, o **auth server** trata o processo de autenticação e de geração de **tokens** para os utilizadores. Em suma, com este servidor conseguimos gerir os utilizadores e as suas informações, bem como gerir a autenticação dos mesmos.

Os servidores acima mencionados serão explicados com maior detalhe numa secção própria. De seguida, segue uma imagem representativa da arquitetura do sistema:

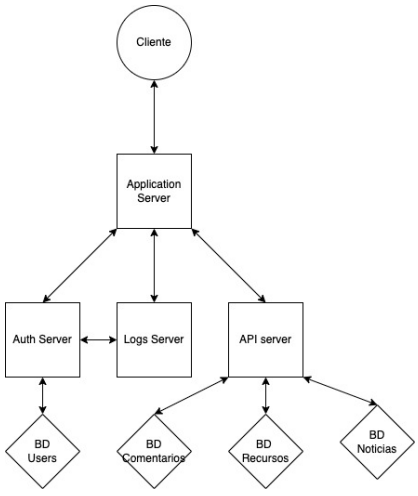

Figura 2.1: Arquitetura do Sistema

Capítulo 3

Utilizadores e permissões

Neste projeto, tal como solicitado, os diferentes níveis dos utilizadores refletem-se em diferentes funcionalidades às quais cada um tem acesso.

3.1 Administrador

O **Administrador** é o nível mais alto de acesso que um utilizador pode ter. Consequentemente, tem acesso a todas as funcionalidades existentes. Em seguida, estas encontram-se listadas:

- Relativamente aos utilizadores, o administrador consegue editar ou remover tanto consumidores como produtores.
- No que concerne aos recursos, o administrador consegue editar, remover, fazer downloads e uploads, comentar e meter like, consultar os ficheiros contidos nos mesmos e consultar os recursos existentes por tipo e por nome.
- Tem acesso às funcionalidades de consultar e remover logs.
- Consegue aceder e editar o seu próprio perfil.
- Relativamente às notícias, tem o poder de as remover e tornar visíveis ou invisíveis.

3.2 Consumidor

O **consumidor** é o nível mais baixo e o tipo de utilizador que tem menos acesso na aplicação desenvolvida. As funcionalidades disponíveis são as seguintes:

- No que concerne aos recursos, consegue consultar os mesmos por nome e tipo, fazer downloads, comentar e meter like e consultar os ficheiros contidos nos mesmos.
- Consegue aceder e editar o seu perfil.
- Tem a funcionalidade de visualizar notícias.

3.3 Produtor

O **produtor** é o nível médio da cadeia de utilizadores do nosso sistema. Este tem menos acessos do que um administrador mas mais acessos do que um consumidor. De seguida, encontram-se as funcionalidades deste tipo de utilizador:

- Relativamente aos recursos, um produtor consegue consultar os mesmos por nome e por tipo, fazer downloads e uploads, comentar e meter like, consultar os ficheiros contidos em cada recursos e consegue editar e remover apenas os recursos submetidos por si.
- Tem a funcionalidade de aceder e editar o seu próprio perfil.
- No que concerne às notícias, consegue visualizar as mesmas.

Capítulo 4

Implementação dos servidores

Neste capítulo são apresentadas as decisões de implementação dos servidores pertencentes à nossa aplicação. Adicionalmente são apresentados alguns detalhes de cada um deles.

4.1 API-Server

Este servidor foi desenvolvido com recurso ao **Node.js** e à framework **Express**. Este fica à escuta na **porta 8003**, e responde com os metadados dos recursos armazenados.

4.1.1 Modelos

Para persistir os dados e metadados da nossa aplicação, decidimos recorrer ao módulo **mongoose**. Foram criados 3 schemas: um para os comentários, um para as notícias e outro para os metadados dos recursos. Com isto é possível estabelecer a conexão com uma base de dados em **MongoDB**, denominada ProjetoRPCW2022.

Os schemas criados permitem-nos ter diversas coleções nessa base de dados e trabalhar sobre as mesmas. As coleções finais criadas através destes schemas são: **recursos**, **noticias** e **comentarios**. Nas figuras seguintes são apresentados os schemas.

```
var recursoSchema = new mongoose.Schema({
    _id: mongoose.Types.ObjectId,
    dataCriacao: String,
    dataSubmissao: String,
    idProdutor: String,
    idSubmissor: String,
    titulo: String,
    tipo: String,
    path: String,
    likes: Number,
    users_liked : [String]
})
```

Figura 4.1: Schema do recurso

```
var noticiaSchema = new mongoose.Schema({
    _id: mongoose.Types.ObjectId,
    nome: String,
    acao: String,
    data: String,
    idRecurso: String,
    visivel: Boolean
})
```

Figura 4.2: Schema da notícia

```
var comentarioSchema = new mongoose.Schema({
    _id: mongoose.Types.ObjectId,
    user: String,
    data: String,
    idRecurso: String,
    texto: String
})
```

Figura 4.3: Schema do comentário

4.1.2 Controladores

Relativamente aos controladores, utilizámos um para cada Schema anteriormente mencionado. Recorrendo também ao módulo **mongoose** e ao respetivo modelo, implementámos um conjunto de queries cruciais para o funcionamento do repositório.

Para os comentários implementamos as seguintes queries: inserção, listagem por id, listagem por id de um recurso e remoção.

Para as notícias implementamos: atualização por nome do autor, inserção, atualização da visibilidade, listagem, listagem por id e remoção.

Por fim, para os recursos implementamos: inserção, atualização, listagem, listagem por tipo, filtragem por expressões regulares, listar pelo id do recurso, remoção, atualização do tipo e do título, listagem por submissor, atualização da contagem de likes, atualização do submissor e atualização da lista de likes.

4.1.3 Roteadores

API [/api]

Este roteador recebe todos os pedidos relacionados com os metadados dos recursos.

Método	Rota	Descrição
PUT	/recursos/atualizarUser	Sempre que é alterado o username de um utilizador, é feito um pedido a esta rota para atualizar o username nas bases de dados do recurso.
GET	/recursos/:rid	Pedido para listar os metadados de um recurso com id igual a rid.
GET	/recursos	Listagem de todos os recursos. Opcionalmente, pode ser pedida uma filtragem por tipo (?tipo=X), de acordo com um padrão proveniente da barra de pesquisa (?search=X) ou por um submissor (?submissor=X).
DELETE	/recursos/:rid	Eliminar um recurso com id igual a rid.
POST	/recursos	Inserção de um recurso.
PUT	/recursos/:rid/atualizarLikes	Atualizar a lista de likes de um recurso.
PUT	/recursos/:rid	Atualização de um recurso. Pode ser atualizado o seu tipo, o seu título, ou ambos.

Tabela 4.1: Rotas do roteador API

Comentários [/comentarios]

Este roteador recebe todos os pedidos de dados dos comentários dos recursos.

Método	Rota	Descrição
GET	/:id	Listagem de todos os comentários do recurso com id igual a id.
POST	/	Inserção de um novo comentário.
DELETE	/:id	Remoção do comentário com id igual a id.

Tabela 4.2: Rotas do roteador dos comentários

Notícias [/noticias]

Este roteador recebe todos os pedidos de dados das notícias.

Método	Rota	Descrição
PUT	/atualizarUser	Sempre que um username de um utilizador é atualizado, é feito um pedido a esta rota para atualizar as ocorrências do username antigo nas notícias.
PUT	/:id	Atualizar a visibilidade de uma notícia.
GET	/	Listagem de todas as notícias.
POST	/	Inserção de uma notícia.
DELETE	/:id	Remoção de uma notícia.

Tabela 4.3: Rotas do roteador das notícias

4.1.4 Proteção das rotas

Com o objetivo de proteger os acessos às rotas dos diferentes roteadores, utilizamos o módulo **jsonwebtoken** para verificar se o token de acesso existe, e se este é válido. Desta forma incluímos uma função (jwt.verify) que nos permite verificar o token, e utilizamos essa função em todas as rotas deste servidor. Adicionalmente criámos uma função para verificar o nível do utilizador nos casos em que era necessário.

4.2 App-Server

Este servidor foi desenvolvido com recurso ao **Node.js** e à framework **Express**. Este fica à escuta na **porta 8001** e é o principal serviço da aplicação desenvolvida. É este servidor que responde a todos os pedidos de visualização de qualquer utilizador, desde que tenha acesso a tais recursos, e que serve de intermediário com os restantes servidores.

4.2.1 Roteadores

Index [/]

Este roteador é o responsável por responder aos pedidos mais básicos do utilizador, nomeadamente o acesso à página principal, o acesso à pagina de registo, o acesso à página de login e o acesso à página de upload.

Recursos [/recursos]

Este roteador recebe todos os pedidos relacionados com os recursos.

Método	Rota	Descrição
POST	/upload	Permite ao utilizador fazer o upload de um recurso para o repositório
GET	/	Devolve ao utilizador todos os recursos existentes na base de dados ou um recurso em específico, de acordo com o seu identificador
GET	/consultaOnline	Permite ao utilizador visualizar um documento presente num recurso numa nova janela
GET	/administrar	Devolve ao administrador uma página com a listagem de todos os recursos existentes onde pode realizar a gestão dos mesmos.
GET	/search	Devolve ao utilizador todos os recursos existentes que correspondam à pesquisa que este realizou
GET	/tipo	Devolve ao utilizador todos os recursos existentes de acordo com o tipo pretendido
GET	/atualizarLikes/:rid	Comunica com a api para atualizar o número de likes de um recurso correspondente ao identificador especificado, podendo aumentar ou diminuir os mesmos
GET	/eliminar:/rid	Comunica com a api para eliminar um dado recurso, de acordo com o seu identificador, da base de dados bem como do file storage.
GET	/editar/:rid	Devolve ao utilizador um página de edição do recurso especificado
POST	/editar/:rid	Atualiza na base de dados de acordo comas edições feitas pelo utilizador
GET	/download/:rid	Permite ao utilizador descarregar um dado recurso presente no sistema de acordo com o seu identificador
POST	/comentar/:rid	Adiciona na base de dados um comentário a um recurso introduzido pelo utilizador, de acordo com o identificador do recurso
GET	/comentarios/eliminar/:rid	Permite ao utilizador eliminar um comentário relativo a um recurso especificado pelo identificador deste último

Tabela 4.4: Rotas do roteador de recursos

User [/users]

Este roteador é responsável por responder a todos os pedidos relativos a utilizadores.

Método	Rota	Descrição
GET	/	Devolve ao administrador todos os utilizadores existentes na base de dados
POST	/perfil	Atualiza o perfil de um utilizador na base de dados
GET	/editarPerfil	Devolve ao utilizador uma página de edição do seu perfil
GET	/perfil	Devolve ao utilizador a página do seu perfil
GET	/eliminar/:username	Permite ao administrador eliminar um utilizador
GET	/editar	Permite ao administrador alterar o nivel de um utilizador
POST	/editar	Atualiza na base de dados o nível de um dado utilizador
POST	/registar	Permite a qualquer utilizador fazer o registo no serviço
POST	/login	Permite a qualquer utilizador efetuar login no serviço
GET	/logout	Permite a qualquer utilizador efetuar o logout do serviço

Tabela 4.5: Rotas do roteador de utilizadores

Noticias [/noticias]

Este roteador é repsonsável pela gestão de notícias.

Método	Rota	Descrição
GET	/editar/:idNoticia	Permmite ao administrador editar a visibilidade de uma dada noticia
GET	/eliminar/:idNoticia	Permite ao administrador eliminar uma dada notícia

Logs [/logs]

Este roteador é responsável pela gestão de logs do serviço.

Método	Rota	Descrição
GET	/	Devolve ao administrador todos os logs existentes
POST	/	Atualiza os logs do serviço
GET	/delete/:id	Permite ao administrador eliminar um log

´

4.2.2 Pacotes de Informação

Sendo um dos mais importantes pontos deste trabalho os pacotes de informação associados aos recursos, consideramos importante mencionar e explicar o método seguido durante o projeto para que este cumprisse com as normas mencionadas no enunciado.

Upload de Recursos

Um produtor, ou administrador, ao efetuar o upload de um recurso deve cumprir com determinados requisitos para que o mesmo seja válido.

Deve conter principalmente dois ficheiros: um com meta informação sobre o recurso que está a ser adicionado ao repositório e outro, o manifesto (RRD-SIP) que deve conter informação que permita comprovar o que está contido no recurso, nomeadamente os ficheiros do mesmo.

Este último deve também comprovar que os ficheiros contidos no recurso são de facto os especificados no manifesto, através do checksum, calculado através do algoritmo especificado no mesmo.

Desta forma, conseguimos garantir que os recursos que sejam adicionados ao respositório são efetivamente válidos.

Download de Recursos

De forma idêntica ao upload, quando fazemos o download de um recurso torna-se necessário reavaliar o mesmo, uma vez que poderá ter sofrido alterações, de forma a manter a integridade dos pacotes de informação e do próprio recurso.

Assim, quando um utilizador deseja descarregar um recurso para uso próprio o mesmo contém os mesmo dois ficheiros que existem aquando do upload, o de metadados e o manifesto mas com os campos atualizados.

4.2.3 File Storage

Estando preparado para lidar com uma enorme quantidade de recursos, torna-se necessário organizar o armazenamento dos mesmos de forma a que o acesso posterior seja facilitado e mais eficiente por parte do nosso serviço.

Assim, aquando do upload do recurso é calculado a *hash*, com base no algoritmo **sha-256**, do título do recurso juntamente com a sua data de criação. Posteriomente é dividido ao meio possibilitando assim a criação de duas pastas com identificações únicas para o recurso submetido.

Quando pretendemos aceder a um recurso, para efetuar download, ou aos ficheiros nele contidos, para efeitos de pré-visualização, basta calcularmos este valor através da meta informação contida na nossa base de dados passando a ter acesso direto ao mesmo.

4.3 Auth-Server

Este servidor foi desenvolvido com recurso ao **Node.js** e à framework **Express**. Este fica à escuta na **porta 8002**, e responde aos pedidos relacionados com a gestão de utilizadores, as suas informações e os seus **json web tokens**, cruciais para aceder às rotas protegidas.

4.3.1 Modelos

Recorrendo mais uma vez ao módulo **mongoose**, e utilizando a mesma base de dados já utilizada no servidor da API de dados, decidimos criar mais um schema para persistir a informação dos utilizadores. A coleção criada para os utilizadores é a **users**. Na figura 4.4 é apresentado o schema.

```
var userSchema = new mongoose.Schema({
    username: String,
    password: String,
    nivel: String
})
```

Figura 4.4: Schema do utilizador

4.3.2 Controladores

Neste servidor tiramos partido de apenas um controlador, relativo aos utilizadores. Utilizámos o módulo **mongoose** e o Schema do utilizador para criar um conjunto de queries necessárias para interagir com a coleção. Estas são: registo de um novo utilizador, listagem de utilizadores ordenando pelo username, listagem de utilizadores por nível, consulta de um utilizador por username, remoção de um utilizador, alteração do seu nível e alteração das suas credenciais.

4.3.3 Roteadores

Autenticação [/auth]

Este é o único roteador deste servidor, e portanto, responde a todos os pedidos dirigidos ao mesmo.

Método	Rota	Descrição
GET	/users/meuPerfil	Devolve a informação relativa ao utilizador atualmente autenticado.
GET	/users/:username	Devolve a informação do utilizador com username igual a username.
GET	/users	Devolve a informação de todos os utilizadores, separando os produtores dos consumidores.
POST	/login	Pedido de login. É feita a autenticação utilizando a estratégia local e gerado um json web token para utilizar nos futuros pedidos.
POST	/registar	Pedido para registar um novo utilizador. No body são enviadas as credenciais (username e password) e também o nível. A password é sempre encriptada com o algoritmo sha256 e só depois é que é armazenada na base de dados. Se o registo for feito através da aplicação, o nível não é especificado no body e o valor predefinido será o de consumidor. Para criar um administrador ou produtor o pedido pode ser feito através do postman.
DELETE	/eliminar	Deve ser especificado um utilizador na querystring. Este será eliminado do sistema.
PUT	/users/editarPerfil/:username	Esta rota é utilizada para editar o perfil do utilizador atualmente autenticado. As suas credenciais são alteradas, e no caso de ter sido alterado o username, são feitas as alterações necessárias nas bases de dados das notícias, dos comentários e dos recursos.
PUT	/users	Através desta rota é alterado o nível de um utilizador. O nível e o username devem ser enviados na query string.

Tabela 4.6: Rotas do roteador da autenticação

4.3.4 Estratégia de autenticação

Para todo o processo de autenticação utilizamos os módulos **passport**, **passport-local**, **session** e **jsonwebtoken**.

Antes de cada pedido, no ficheiro **app.js**, definimos que iria ser utilizada uma sessão, com a secret **ProjetoRPCW2022**. Posteriormente definimos a estratégia local, implementamos a serialização e desserialização do utilizador, e por fim inicializámos as funcionalidades do módulo passport. Na figura 4.5 está apresentada a estratégia local.

No roteador, utilizámos uma função para verificar o token em todos os pedidos que necessitassem de autenticação. No login ocorre um caso especial visto que utilizamos a estratégia local para validar as credenciais e só depois disso é que geramos o **json web token**, com a validade de 1 dia.

```
app.use(session({
  secret: 'ProjetoRPCW2022',
  resave: true,
  saveUninitialized: true
}))

// Configuração da estratégia local
passport.use(new LocalStrategy(
  {usernameField: 'username'}, (username, password, done) => {
    password_encriptada = createHash('sha256').update(password).digest('hex');
    // console.log(password_encriptada)
    User.consultarUtilizador(username)
      .then(dados => {
        const user = dados
        // console.log(dados)
        if(!user) { return done(null, false, {message: 'Utilizador inexistente!\n'})}
        if(password_encriptada != user.password) { return done(null, false, {message: 'Credenciais inválidas!\n'})}
        return done(null, user)
      })
      .catch(e => done(e))
    })
)
```

Figura 4.5: Estratégia local

4.4 Log-Server

Este servidor fica à escuta recorrendo ao módulo **json-server**, e desta forma os seus dados podem ser persistidos num ficheiro JSON, denominado **logs.json**. Quando este servidor é inicializado deve ficar à espera de pedidos na **porta 8004**.

Todas as ações importantes de gestão de recursos, utilizadores, comentários, likes e notícias são armazenadas nos logs, bem como quando alguém faz login e logout. Além disso são também listadas as ações de consulta de recursos, identificando sempre o autor das ações.

Capítulo 5

Resultados

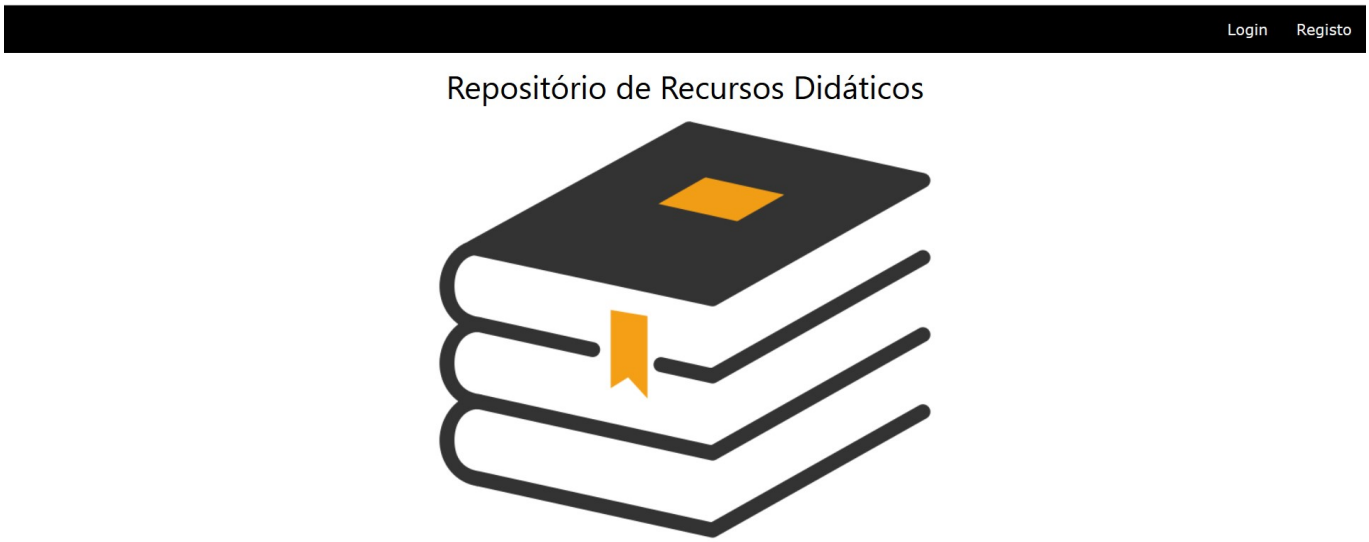

Figura 5.1: Página inicial

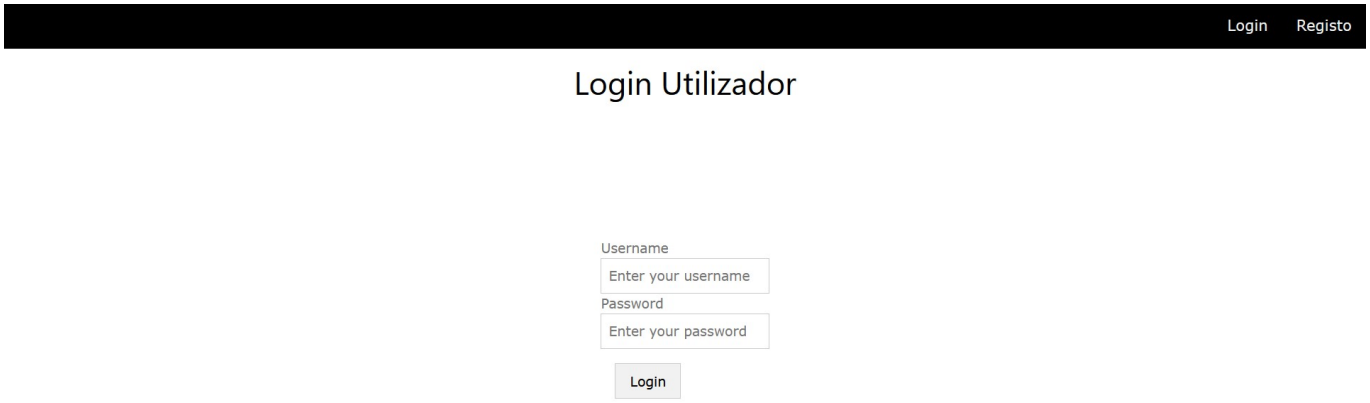

Figura 5.2: Página de login

Figura 5.3: Homepage do admin

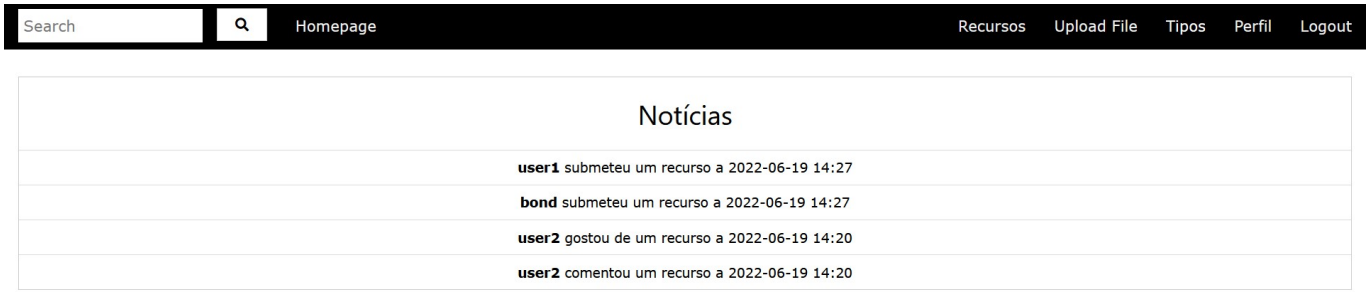

Figura 5.4: Homepage do produtor e consumidor

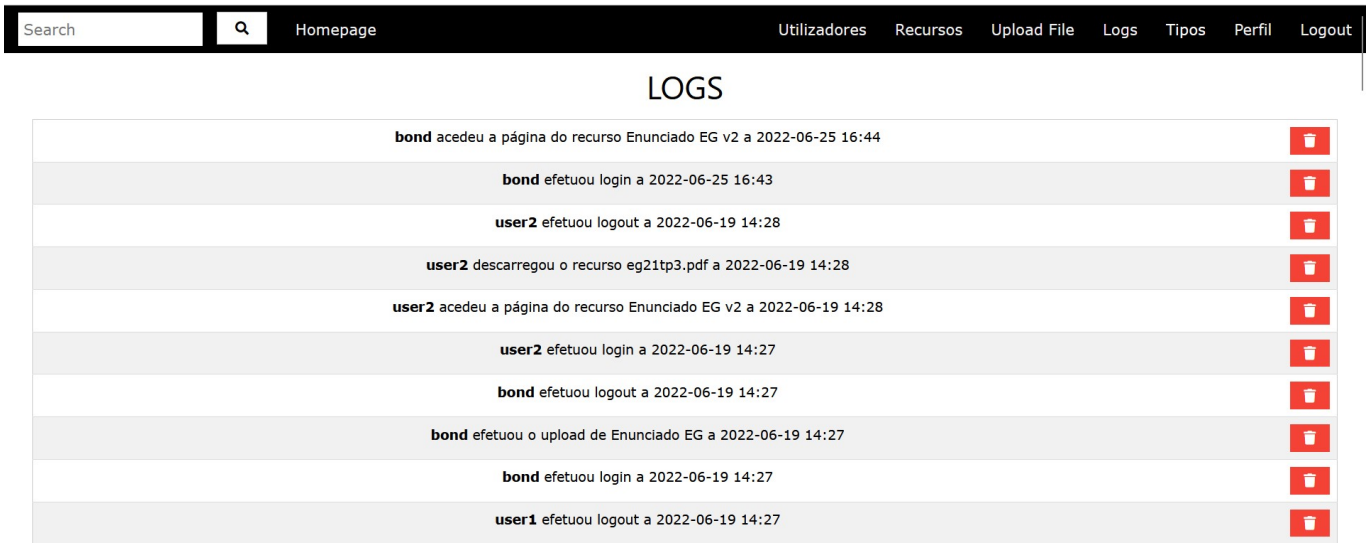

Figura 5.5: Página de logs

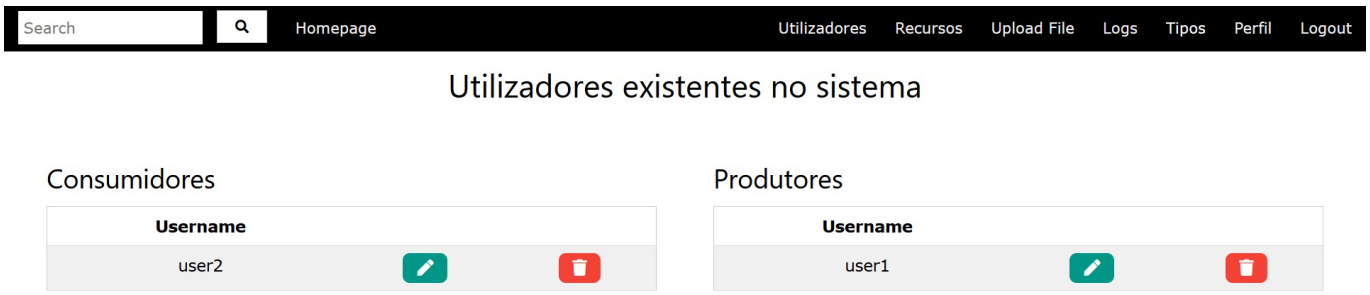

Figura 5.6: Página dos utilizadores

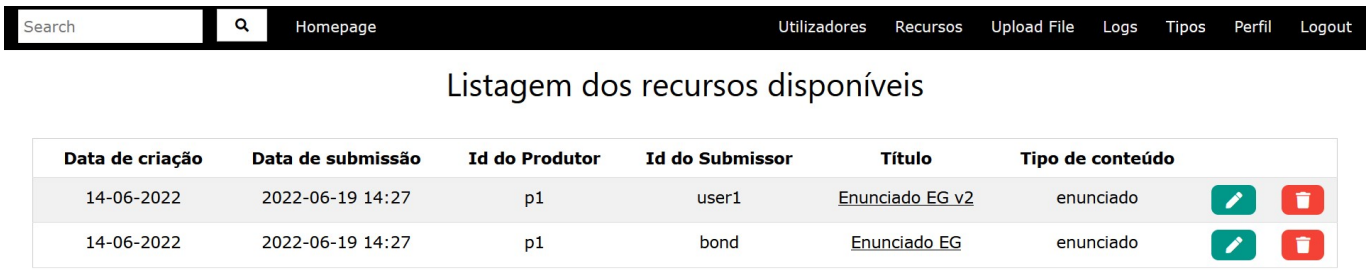

Figura 5.7: Página de recursos (ponto de vista do admin)

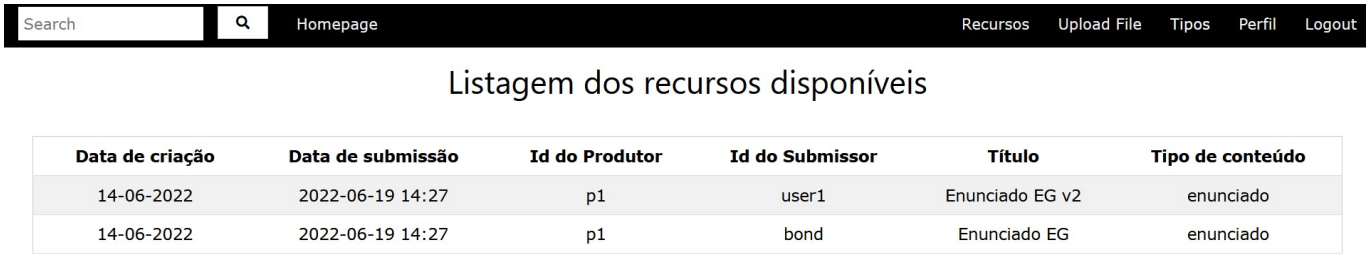

Figura 5.8: Página de recursos (ponto de vista dos outros utilizadores)

Figura 5.9: Página de um recurso

Figura 5.10: Página de upload

Figura 5.11: Perfil do utilizador

Figura 5.12: Editar perfil

Capítulo 6

Conclusão

O presente documento visa apresentar de forma sucinta e concreta o projeto desenvolvido, desde a formulação do problema até à implementação final e respetivos resultados apresentados, tendo sempre como base a explicação do raciocínio utilizado para a resolução do problema em questão.

Revendo todo o trabalho feito até aqui, o projeto desenvolvido encontra-se completo e a equipa acredita que este está ao nível que pretendia desde o seu início. Todos os resultados citados como objetivo no enunciado foram alcançados e realizados sempre da forma que se pensou ser a mais correta. Porém, gostaríamos de salientar apenas que pretendiamos ter obtido mais sucesso através de duas tarefas que acabamos por não concretizar, sendo elas a criação de estatísticas de utilização da aplicação web e o respetivo *deployment* da aplicação através de *docker-compose*.

No âmbito geral, o grupo sente que desenvolveu o trabalho prático da melhor forma possível e conseguiu alcançar a grande maioria dos objetivos propostos no mesmo.