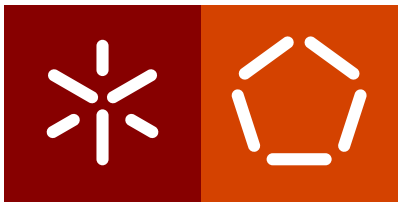
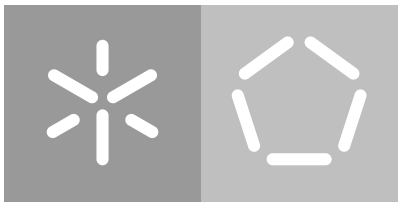


Universidade do Minho
Escola de Engenharia
Departamento de Informática

António Alexandre Carvalho Lindo

CLAV: Gestão de Backups e Importação de Dados

Janeiro, 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

António Alexandre Carvalho Lindo

CLAV: Gestão de Backups e Importação de Dados

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Trabalho realizado sobre a orientação do professor
José Carlos Leite Ramalho

Janeiro, 2022

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

_____

AGRADECIMENTOS

Gostaria de agradecer ao meu orientador José Carlos Ramalho por todo o apoio e orientação durante a execução desta dissertação e também pela oportunidade de trablhar num projeto muito interessante e no qual aprendi muito. Também gostaria de agradecer a todos os meus amigos, em especial Pedro Parente e Nuno Cunha por me terem ajudado e apoiado durante estes 5 anos de curso. Quero, ainda, agradecer aos meus pais por tudo o que fizeram por mim durante estes anos, pelo apoio constante e por todos os sacrifícios. Por fim, gostaria também de agradecer a todos os meus familiares mais próximos.

RESUMO

O CLAV é um projeto nacional financiado pelo Simplex. O objetivo deste projeto é classificar e avaliar toda a documentação circulante na Administração Pública portuguesa. Desta forma, as entidades públicas disporão de uma ferramenta que possibilita a identificação da documentação que deve ser eliminada ou arquivada. No entanto, como em todas as plataformas, podem ocorrer imprevistos que resultem em perda de informação.

Nesta dissertação foi criada uma ferramenta web externa ao CLAV, que permita a execução de backups e importação de informação na plataforma, a fim de ser possível o armazenamento da informação em volumes externos ao CLAV de modo a esta informação poder voltar a ser recolocada no CLAV mais tarde. Para a criação dos pacotes de backup, recorreu-se a formatos standard de armazenamento de informação.

Palavras-Chave: *Backup*, Importação, CLAV

ABSTRACT

CLAV is a national project funded by Simplex. The objective of this project is to classify and evaluate all documents circulating in the Portuguese Public Administration. In this way, public entities will have a tool that makes it possible to identify the documentation that must be eliminated or archived. However, as with all platforms, unforeseen events may occur and result in loss of information.

In this dissertation, a web application external to CLAV was created, which allows to execute backups and import information on the platform, in order to be able to store the information in volumes external to CLAV so that this information can be replaced in CLAV later. To create the backup packages, it was used standard information storage formats.

Keywords: Backup, Import, CLAV

CONTEÚDO

LISTA DE FIGURAS

SIGLAS

A

AIP Archival Information Package.

API Application Programming Interface.

C

CCSDS Consultative Committee for Space Data Systems.

CLAV Classificação e Avaliação da Informação Pública.

D

DIP Dissemination Information Package.

O

OAIS Open Archival Information System.

S

SIP Submission Information Package.

1

INTRODUÇÃO

Atualmente, é cada vez mais comum governos e organizações definirem políticas e estratégias com o objetivo de disponibilizar dados ao público nos domínios da ciência e da Administração Pública. Neste contexto, também a Administração Pública portuguesa tem promovido medidas, de forma a digitalizar os processos, promovendo a otimização desses mesmos processos, a modernização de procedimentos administrativos e a redução do consumo de papel. Tendo presente estes objetivos, a DGLAB (Direção-Geral do Livro, dos Arquivos e das Bibliotecas) propôs a criação de uma Lista Consolidada para a classificação e avaliação da informação pública, que também serviria de referencial para a construção normalizada dos planos de classificação e tabelas de seleção das entidades que executam funções do Estado.

De modo a desenvolver esta iniciativa, a DGLAB, em colaboração com a Universidade do Minho (e financiada pelo *Simplex*), procedeu ao desenvolvimento do CLAV (Arquivo Digital: Plataforma modular de classificação e avaliação da informação pública) (Lourenço et al., 2019).

A plataforma CLAV disponibiliza a Lista Consolidada (Viegas and Lourenço.) associada a um catálogo de legislação e de organismos. A Lista Consolidada é utilizada como referência para a construção normalizada dos planos de classificação e tabelas de seleção das entidades que executam funções do Estado. Esta informação é disponibilizada em formato aberto para a integração nos sistemas de informação das entidades, promovendo a interoperabilidade através da utilização de uma linguagem comum (Lista Consolidada) usada no registo, na classificação e na avaliação de informação pública. A plataforma também viabiliza a desmaterialização dos procedimentos associados à elaboração de tabelas de seleção, tendo como base a Lista Consolidada e o controlo de eliminação e arquivamento da informação pública através da integração das tabelas de seleção (Filipa Carvalho and Lourenço.) nos sistemas de informação das entidades, alertando-as quando determinado documento deve ser arquivado ou eliminado.

1.1 MOTIVAÇÃO

Hoje em dia, o armazenamento de dados e informação é habitualmente realizado em plataformas digitais. A digitalização deste processo deve-se a várias fatores, a saber: maior facilidade na procura de informação e no seu acesso; otimização do tempo e claro, a redução do uso de papel, não só por motivos ecológicos, mas também pelo espaço físico que ocupa. Outra das vantagens é o facto de a informação armazenada em papel poder degradar-se com o tempo ou mesmo ser perdida devido a imprevistos. Assim sendo, assiste-se, cada vez mais, ao incremento da digitalização do armazenamento de vários tipos de dados. No entanto, apesar das plataformas digitais oferecerem uma maior segurança em relação à conservação dos dados, podem sempre ocorrer imprevistos que causem a perda dos mesmos, desde problemas nas máquinas que os guardam a ataques informáticos. Estas perdas têm consequências graves, uma vez que, informação importante é perdida e não pode ser recuperada. Os dados são a base deste tipo de plataformas e a sua perda pode afetar gravemente o funcionamento da plataforma e os seus utilizadores. A plataforma CLAV armazena vasta informação, no entanto, não tem um sistema que permita a execução de *backups* de forma organizada e rápida. Desta forma, e tendo em conta os aspetos acima mencionados, esta dissertação tem como objetivo a criação de uma aplicação *web* que permita aos utilizadores da plataforma CLAV a execução de *backups* da informação presente na plataforma sempre que entenderem, de modo a que seja possível armazenarem os dados presentes na plataforma na *data* em que um *backup* é feito. O processo contrário seria também possível, ou seja, utilizar esses mesmos *backups* para repor dados na plataforma, em caso de perda.

1.2 OBJETIVOS

O objetivo desta dissertação é a implementação de uma aplicação *web* que permita a execução de *backups* e a importação de informação na plataforma. Para alcançar este objetivo será necessário:

- Criar uma política de *backups* e recuperação de toda a informação da plataforma CLAV;
- Especificar um formato para *backup* e importação, baseado em normas internacionais;
- Criar uma política de importação de informação para a plataforma CLAV, através de *backups* já executados;
- Criar uma aplicação que implementa a política especificada e assiste o utilizador nessa ação.

1.3 QUESTÕES DE INVESTIGAÇÃO

No desenvolvimento desta dissertação levantaram-se algumas questões, tais como: que formato utilizar para os *backups* a gerar, que meta-informação apresentar e ainda como permitir a um utilizador inserir um pacote na plataforma CLAV, de modo a importar o seu conteúdo.

Todas as respostas a estas questões estão presentes na conclusão desta dissertação.

1.4 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação inicia no capítulo 2, com a apresentação dos conceitos e tecnologias já existentes, relativos ao armazenamento e transferência de informação.

No capítulo 3, é relatada a abordagem inicial ao problema e apresentada uma breve explicação da arquitetura planeada para a aplicação. No capítulo 4, é descrito todo o funcionamento do processo de execução de *backups*. O capítulo 5 descreve o funcionamento do processo de importação de dados e o capítulo 6 introduz a interface *web* da aplicação e a sua forma de utilização. No capítulo 7, é explicado como é feito o processo de *deployment* e instalação da aplicação através do *Docker*.

Por fim, no capítulo 8 apresentam-se as principais conclusões e respostas às questões de investigação.

2

ESTADO DA ARTE

Desde há muito tempo que se assiste ao incremento da digitalização de vários tipos de processos, entre os quais, o armazenamento de informação. A informação armazenada pode ter de ser transferida entre várias máquinas e sistemas por diversas razões, desde *backups* para evitar as consequências das perdas de dados, a transferências para acesso da mesma informação em outras máquinas. No entanto, estes processos podem não resultar devido a falhas humanas, de rede ou de máquina, resultando em informação corrompida. Devido a este facto, foi necessário criar normas e modelos de referência, de modo a permitir consistência e persistência nos processos de transferência e armazenamento de dados.

2.1 MODELO OAIS

O modelo OAIS (*Open Archival Information System*) é um modelo de referência para armazenamento de longa duração de informação digital. Este modelo começou a ser desenvolvido pelo CCSDS (*Consultative Committee for Space Data Systems*) com vista a criar uma referência para todos os processos relativos a armazenamento e transferências de dados, derivados de missões espaciais em arquivos. Este modelo foi aprovado em 2002 como ISO *International Standard* 14721, tendo sido depois atualizado em 2012, passando a designar-se ISO 14721:2012. Um arquivo que siga este modelo tem como principais funções a ingestão, planeamento de preservação, armazenamento, acesso, gestão e administração. Este tipo de arquivo opera num abiente constituído essencialmente por 3 entidades: Gestão (*Management*), Produtor (*Producer*) e Consumidor (*Consumer*) (Lavoie, 2014).

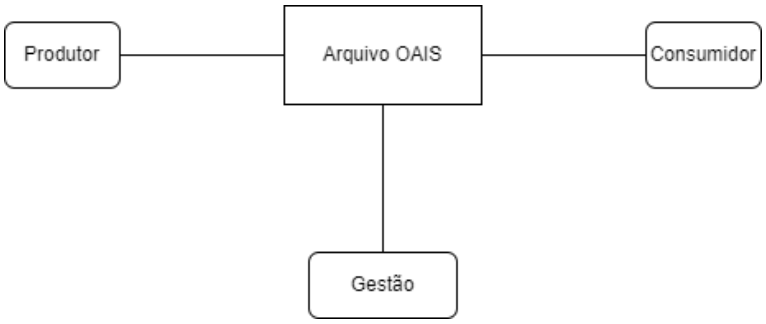


Figura 1: Ambiente de funcionamento de um arquivo OAIS

A informação flui dentro de um arquivo OAIS através de pacotes, existindo 3 tipologias :

- **SIP (*Submission Information Package*):** Pacote submetido por um produtor para o arquivo ingerir;
- **AIP (*Archival Information Package*):** Pacote que o arquivo guarda, de modo a ser preservado. Este pacote é gerado a partir de um SIP;
- **DIP (*Dissemination Information Package*):** Pacote gerado a partir de um ou mais AIP, que é recebido por um consumidor após um pedido ao arquivo.

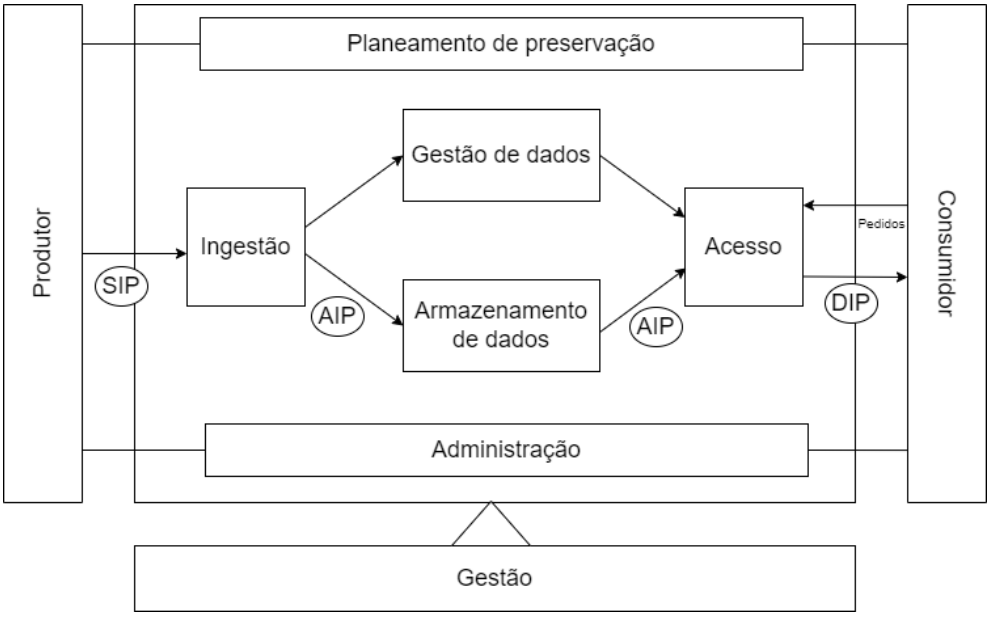


Figura 2: Arquitetura e funcionamento de um arquivo OAIS

Tendo o modelo OAIS em mente, foram criados vários tipos de pacotes SIP, tais como o BagIt e o E-ARK SIP.

2.2 BAGIT

Em 2009, a *Library of Congress* dos Estados Unidos da América, publicou um vídeo (Library of Congress) em que explicava detalhes sobre uma norma criada para este efeito, em colaboração com a *California Digital Library*, designada *BagIt*. Esta norma foi primeiramente utilizada em 2007, quando a *California Digital Library* precisou de transferir vários terabytes de arquivos para a *Library of Congress*, tendo ajudado na uniformização do processo e na verficação de erros. Tal como um envelope é utilizado para transferir informação entre várias localizações, o *BagIt* é um pacote utilizado para o mesmo efeito, mas entre máquinas. Cada *BagIt* (denominado de *bag*) tem essencialmente a seguinte estrutura:

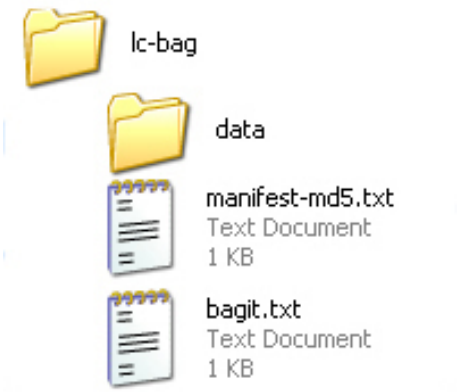


Figura 3: Estrutura de uma *bag*

Uma *bag* pode assumir qualquer designação, devendo obrigatoriamente apresentar:

- **Uma diretoria de nome "data":** Contém toda a informação a transferir, desde diretorias a todo o tipo de ficheiros;
- **Um manifesto:** Ficheiro de texto que lista os ficheiros existentes na *bag* e os seus respetivos *checksums* calculados a partir de um algoritmo que deve estar incluído no nome deste ficheiro (sha256, md5 etc). No caso da figura 1, o algoritmo utilizado foi o md5.
- **Uma declaração da *bag*:** Ficheiro de texto que contém meta-informação da *bag* em si, tal como a sua versão e *enconding*.

Estes três elementos são de presença obrigatória numa *bag*, no entanto, a norma permite a existência de outros ficheiros opcionais, entre os quais um ficheiro *bag-info.txt*. Este ficheiro pode conter informação relativa ao criador da *bag* como nome, organização, etc, e ainda tamanho e data de criação da *bag*.

2.3 E-ARK SIP

Um outro formato, também utilizado, é o E-ARK SIP. Este formato foi desenvolvido entre 2014 e 2017, através do projeto *E-ARK*, sendo posteriomente publicadas as suas especificações (Bredenberg et al., 2018). A estrutura de um E-ARK SIP é a seguinte:

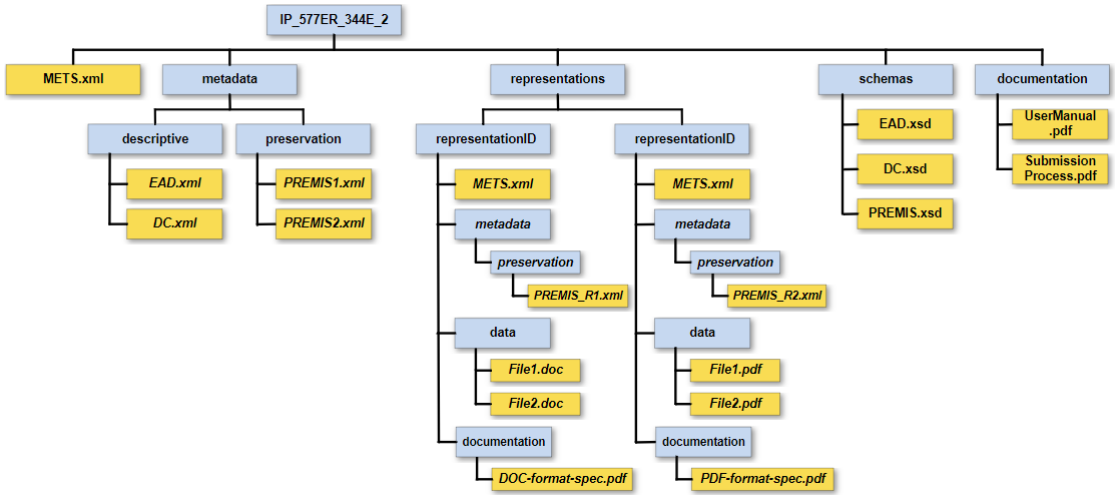


Figura 4: Estrutura do E-ARK SIP

O E-ARK SIP é mais complexo que o *BagIt*, contendo mais diretorias e ficheiros, que têm as seguintes funções:

- **Ficheiro *METS.xml*:** Ficheiro que guarda informação relativa à estrutura do pacote E-ARK SIP, que o permita identificar como um mapa de ficheiros, metadata sobre o criador, a forma como os ficheiros foram criados e guardados, entre outros;
- **Diretoria metadata:** Contém metadata, podendo esta dividir-se por diretorias de acordo com o seu tipo (descriptive, preservation etc);
- **Diretoria schemas:** Contém ficheiros *xml* com *schemas* relativos à estrutura da metadata e informação adicional;
- **Diretoria documentation:** Armazena ficheiros de documentação;
- **Representations:** Contém uma ou mais representações (cada uma, numa diretoria diferente). Cada representação contém os dados propriamente ditos, podendo ainda conter metadata e documentação específicas dessa mesma representação. Em cada diretoria de representação, pode igualmente existir um ficheiro *METS.xml* que guardará informação relativa à estrutura da representação.

2.4 RESUMO

Neste capítulo, é descrito o atual estado da arte do armazenamento de informação, sendo abordado o modelo OASIS e também dois tipos de modelos para armazenamento de informação que seguem o modelo referido: E-ARK SIP e BagIt. Para os pacotes de *backup* gerados, acabou por ser utilizado o modelo BagIt, tal como é referido no capítulo seguinte.

3

ABORDAGEM PROPOSTA

Como referido anteriormente, o objetivo desta dissertação foi a criação de uma aplicação *web* que permita fazer *backups* e recuperação de informação da plataforma CLAV. Para alcançar este objetivo, foi implementado um *back-end* que terá as seguintes funções:

- ***Backups***: Esta parte do *back-end* é responsável por aceder à informação presente na plataforma CLAV, requisitada pelo utilizador através da API do CLAV (Martins, 2020), processá-la e criar um *backup* da mesma, que pode depois ser transnferido;
- **Importação**: Esta parte tem como função receber *backups* já criados e inserir a sua informação na plataforma CLAV, da forma que o utilizador pretender, isto é, inserir substituíndo toda a informação presente na plataforma CLAV, apenas inserir a informação que ainda não está presente na plataforma ou simplesmente inserir toda a informação presente no pacote de *backup*. Isto, mais uma vez, utilizando a API do CLAV.

Em termos de formato dos pacotes de *backup*, foi definida a utilização do formato *BagIt* por ser mais simples em termos de implementação, bem como mais intuitivo e de fácil perceção para os utilizadores. Os pacotes criados na plataforma terão o formato já referenciado em 2.2, tendo ainda sido acrescentado um ficheiro html com estatísticas relativas ao pacote gerado.

Foi, também, implementado um *interface web* que permitirá ao utilizador executar todos os processos anteriomente descritos e gerir os *backups* que já foram efetuados (transferir ou eliminar). A arquitetura da aplicação será a seguinte:

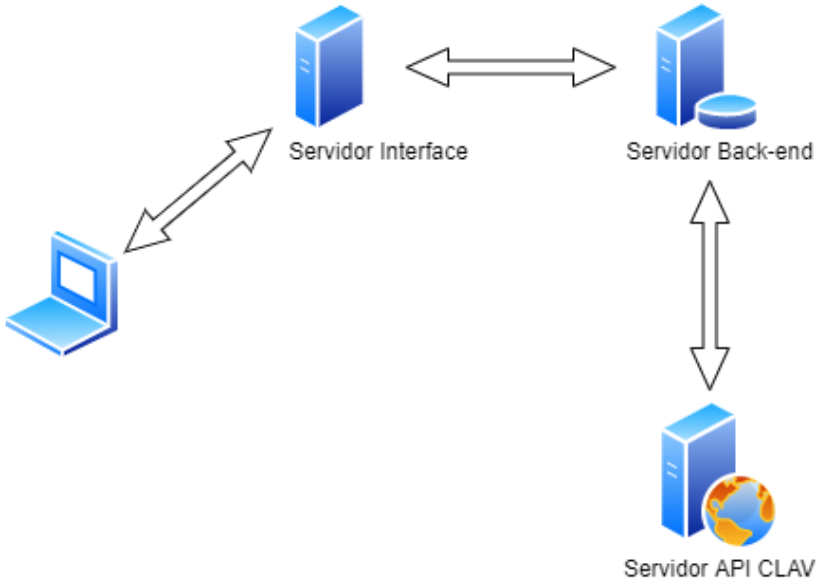


Figura 5: Arquitetura Aplicacional

A aplicação *Web* é implementada em JavaScript. Foi utilizado Node.js para excutar o código em *runtime* e ainda a *framework Express*, que oferece várias funcionalidades úteis para facilitar a implementação de aplicações *web*. Quanto à interface, foi utilizado VueJS, para gerar o html.

3.1 RESUMO

Neste capítulo, é descrita a abordagem proposta para alcançar o objetivo da dissertação, sendo referidas as tecnologias utilizadas, as funcionalidades, a arquitetura aplicacional e a escolha do modelo de armazenamento de informação a utilizar para os pacotes gerados.

4

BACKUPS

A aplicação permite que um utilizador com conta na plataforma CLAV, consiga fazer *backup* de informação presente na plataforma enunciada. O utilizador pode escolher qual a informação que pretende fazer *backup*, selecionando apenas a(s) pretendida(s). Após esta seleção, e validadas as credenciais do utilizador no CLAV, é gerado um pacote com o formato Bagit (descrito em 2.2), contendo os dados pedidos. Este pacote apresenta-se em formato zip.

4.1 TIPOS DE INFORMAÇÃO PARA BACKUP

Como já foi mencionado anteriormente, a plataforma CLAV contem vários tipos de informação. Foi feita uma análise de modo a selecionar os tipos de informação mais cruciais para *backup*, tendo sido selecionados os seguintes:

- **Classes**: Estrutura hierárquica de classes que representam as funções e atividades executadas pela Administração Pública. Os processos de negócio são representados como classes de 3º nível, enquadrados em funções (classes de 1º nível) e subfunções (2º nível). As classes são constituídas por elementos informativos, agrupados por zonas, que as identificam e descrevem. Nas classes de 3º e 4º nível (subdivisão do processo de negócio para efeitos de avaliação), estes elementos destinam-se também a contextualizar e avaliar a informação;
- **Entidades**: Entidades públicas que intervêm nos processos de negócio (classes de 3º nível) da Lista Consolidada. Podem integrar uma ou mais tipologias de entidades;
- **Tipologias**: Forma de agrupamento de entidades que intervêm nos processos de negócio (classes de 3º nível) da Lista Consolidada;
- **Legislações**: Legislações que regulam os processos de negócio e enquadram os respetivos prazos de conservação administrativa (PCA) e destino final (DF);
- **Notícias**: Notícias presentes na plataforma CLAV;

- **Utilizadores**: Utilizadores presentes na plataforma CLAV;
- **Pendentes**: Trabalhos em curso guardados para mais tarde terem continuidade: criação e alteração de instâncias;
- **Documentação de Apoio**: Documentos de apoio presentes na plataforma CLAV;
- **Pedidos**: Pedidos de alteração ou de criação de novas instâncias que deram entrada na plataforma;
- **Invariantes**: Conjunto de invariantes que testam situações de erro e identificam os PNs onde estas ocorrem;
- **Termos de Índice**: Termos que detalham o âmbito de aplicação dos processos de negócio e apoiam a recuperação da informação;
- **PGD e PDGLC**: Portarias de Gestão de Documentos;
- **RADA OLD**: Antigos Relatórios de Avaliação de Documentação Acumulada não inseridos pela plataforma.

4.2 Introdução à API

De modo a implementar as funcionalidades em cima descritas, foi criada uma API. Esta API contem várias rotas que serão enumeradas e introduzidas nas subsecções seguintes. Ao longo deste e dos próximos capítulos, são explicadas com mais detalhe todas as rotas e a forma como são utilizadas.

4.2.1 *POST /backup*

Esta rota é utilizada para a execução de *backups*. Pode ser utilizada das seguintes três formas:

- ***Query string token e nome de utilizador no body***: Este modo de utilização requer uma *query string* com um *token* de autenticação do CLAV e ainda um parâmetro de nome *username* no *body* do pedido com o nome do utilizador.
- ***Query string apikey***: Neste caso, é apenas requerido uma *query string* com uma apikey válida do CLAV;
- **Nome de utilizador e *password* no *body*** Neste modo, é requerido um parâmetro de nome *username* e outro de nome *password* no *body* do pedido que irão conter, respetivamente, o *email* e *password* de um utilizador do CLAV.

O facto de existirem três maneiras diferentes de utilização da rota, deve-se ao facto de haverem também três tipos possíveis de autenticação, que são referidos em 4.3. Em cada uma destas opções, é obrigatória a existência de um parâmetro no *body* do pedido de nome col que será uma lista que contem as coleções de informação a fazer *backup*.

4.2.2 *POST /bagit*

Esta rota é utilizada para guardar as informações dos pacotes de *backup* gerados numa base de dados de modo a estes poderem ser futuramente geridos. A rota necessita apenas de dois parâmetros no *body* do pedido:

- ***username***: Contem o *email* de um utilizador do CLAV.
- ***password***: Contem a *password* de um utilizador do CLAV;

4.2.3 *GET /bagit/download/:bag*

Esta rota permite transferir um pacote de *backup* que é passado como parâmetro na rota. Esta rota requer autenticação, sendo necessários dois parâmetros no *body* do pedido:

- ***username***: Contem o *email* de um utilizador do CLAV.
- ***password***: Contem a *password* de um utilizador do CLAV;

4.2.4 *DELETE /bagit/:nome*

Esta rota permite eliminar da base de dados um pacote de *backup* cujo nome é passado como parâmetro na rota. Esta rota requer autenticação, sendo necessários dois parâmetros no *body* do pedido:

- ***username***: Contem o *email* de um utilizador do CLAV.
- ***password***: Contem a *password* de um utilizador do CLAV;

4.2.5 *POST /import*

Esta rota permite a importação de informação presente em pacotes de *backup* para o CLAV. O pedido requer o seguinte:

- ***File***: Ficheiro a importar (que será o pacote).
- ***Query string token***: *Token* de autenticação do CLAV;

- ***Query string opcao***: Esta *query string* indica o tipo de importação a executar e pode conter os valores: total, overwrite ou adicionar.;

4.3 TIPOS DE AUTENTICAÇÃO

De modo a implementar esta funcionalidade, foi criada uma rota */backup* (4.2.1). Esta rota é do tipo *Post*, que requer autenticação para poder ser utilizada. Os tipos de autenticação permitidos são com *token*, apikey ou com *email* e *password* do CLAV.

4.3.1 *Autenticação com token*

Neste tipo de autenticação, o pedido à rota POST */backup* requer uma *query string* com o *token*, a lista de informação a fazer *backup* e ainda o nome do utilizador CLAV que efetuou este pedido. O nome é requerido, de modo a que possa ser guardado na base de dados o criador deste pacote. Isto porque, não é possível aceder ao nome do utilizador a que um dado *token* pertence. Com este tipo de autenticação, é possível fazer *backup* de todos os tipos de informação disponíveis. Este tipo de autenticação é utilizado no *frontend* da aplicação, como se verá mais adiante.

4.3.2 *Autenticação com email e password*

Neste tipo de autenticação, é requerido no *body* do pedido, o *email* e a *password* do utilizador CLAV, para além da lista da informação. Através destas credenciais é gerado um *token* que será utilizado para fazer os pedidos à API do CLAV. Este tipo de autenticação é utilizado, por exemplo, pelo sistema N8N (descrito em 4.5), que executa *backups* periodicamente e automaticamente, tendo sempre de fazer a autenticação para poder obter um *token*.

4.3.3 *Autenticação com apikey*

Neste tipo de autenticação, é requerido a lista de informação para *backup* e uma *query string* com a *apikey* do CLAV. A partir desta, são feitos os pedidos. No entanto, uma *apikey* não permite aceder a certos tipos de informação do CLAV, estando apenas disponíveis os seguintes: entidades, tipologias, classes, lesgislações, documentação apoio, radas, vocabulários, pgd, pgdLC e termos índice.

4.4 CRIAÇÃO E ESTRUTURA DO PACOTE

Quando é feito um pedido para esta rota, a aplicação começa por criar uma diretoria no *fileStore* com um nome gerado automaticamente, que irá conter o *BagIt* com a informação. Dentro dessa diretoria, são gerados os ficheiros que fazem parte do formato *BagIt* (bag-info.txt, bagit.txt e o manifesto), um ficheiro *html* de estatisticas e ainda uma outra diretoria de nome *data*, que irá conter outras diretorias, cada uma com a respetiva informação.

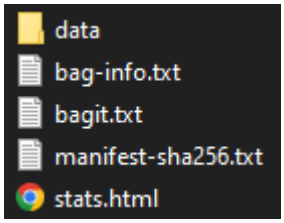


Figura 6: Formato do BagIt gerado pela aplicação

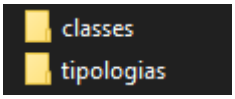


Figura 7: Exemplo do interior da diretoria data

4.4.1 *Ficheiro bag-info.txt*

Este ficheiro contém meta-informação relativa ao *BagIt* gerado, tais como a sua data de criação, tamanho e ainda qualquer outro tipo de informação que se pretenda adicionar, por exemplo, nome da organização responsável, do programador, etc. O tamanho e data de criação são gerados imediatamente após o fim do processo de construção do pacote. Quanto às outras, podem ser adicionadas ou alteradas manualmente no ficheiro bagIt.js presente na diretoria *package-generators*. Na função de contrução de um *Bagit* chamada buildBag, encontra-se um objeto chamado *info*, que tem os seguintes valores pré definidos:

```
1 {'Contact-Name': 'Antonio Lindo', 'Organization-Address': 'Universidade do Minho'
    }
```

Para alterar ou adicionar mais informação, basta adicionar a mesma no objeto.

4.4.2 *Ficheiro bagit.txt*

Este ficheiro contém alguma meta informação relativa à versão do *BagIt* e do *enconding*. Tal como o ficheiro acima, é também gerado durante a construção inicial do pacote.

4.4.3 *Ficheiro Manifesto*

Este ficheiro de nome manifest-sha256.txt, é um ficheiro que contém os *checksums* de todos os ficheiros de informação presentes no pacote. Tal como o nome descreve, utiliza o algoritmo *sha256* para calcular os *checksums*. À medida que os ficheiros de informação vão sendo processados, é escrito no ficheiro uma linha que contém o *checksum* e ainda o *path* de cada ficheiro:

```
1 b4e9a5452d03c6ec1b8d799e5ea499cd16c85459f03159009b23507373a0b1ba data/classes/
    classes.json
e3b1e7d25475e71f237fac91ac1b69bab38cae292d9407f02783866dbc6e2205 data/tipologias/
    tipologias.json
```

4.4.4 *Ficheiro de Estatisticas*

Este ficheiro *html* de nome stats.html contém estatísticas relativas à quantidade de informação que se encontra no pacote, por exemplo, número de utilizadores, de legislações, etc, conforme a informação requisitada para *backup*.

4.4.5 *Processamento da informação*

Estando o pacote inicial gerado, a aplicação começa a processar a informação requisitada para *backup* presente numa lista no *body* do pedido. Esta informação é processada uma a uma. A aplicação inicia este processo verificando se a informação pedida é válida e existe no CLAV. Depois, em caso positivo, são feitos pedidos à API do CLAV, de modo a obter os ficheiros com a informação, utilizando o *token* ou a *apikey*. Caso a informação seja inválida, a aplicação indica este facto no ficheiro de estatísticas e passa ao próximo elemento da lista. Posteriormente, é criada uma diretoria na diretoria *data* com o nome de cada tipo de informação, onde estas serão guardadas (por exemplo, a informação relativa às classes fica guardada numa diretoria de nome Classes).

Legislações, Tipologias, Entidades, Utilizadores, Notícias, Termos Índice, Invariantes e Pedidos

A informação relativa a estes encontra-se num único ficheiro JSON criado, onde são escritos os dados resultantes do pedido à API do CLAV. Este ficheiro será lido para contabilizar o número de objetos que contém, de modo a colocar nas estatísticas. Ao mesmo tempo, é gerado o seu respetivo *checksum*, escrito no manifesto, sendo por fim tudo guardado na diretoria criada para o efeito.

Documentação de Apoio

No caso da documentação de apoio, é feito um pedido ao CLAV de modo a obter um ficheiro JSON com toda a documentação existente. Neste ficheiro, estão guardados os nomes dos pdfs, *links*, entre outra informação. Nos casos em que existe um *link* direto para o pdf, é feito um *download* direto do ficheiro, utilizando o módulo *DownloaderHelper* do *NodeJS*. Quando algum documento não possui *link*, é feito um pedido à API do CLAV de modo a obter o conteúdo do pdf em causa. Posteriormente, é criado um ficheiro pdf em branco com o nome do documento em causa e o conteúdo obtido com a chamada à API, é nele escrito. Estas operações são feitas documento a documento, sendo como sempre calculado o *checksum* de cada ficheiro e escrito no manifesto e calculado o número de documentos para as estatísticas. Todos os documentos são guardados numa diretoria de nome ficheiros.

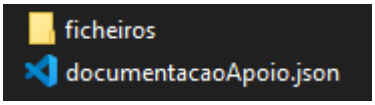


Figura 8: Conteúdo da diretoria de documentação apoio

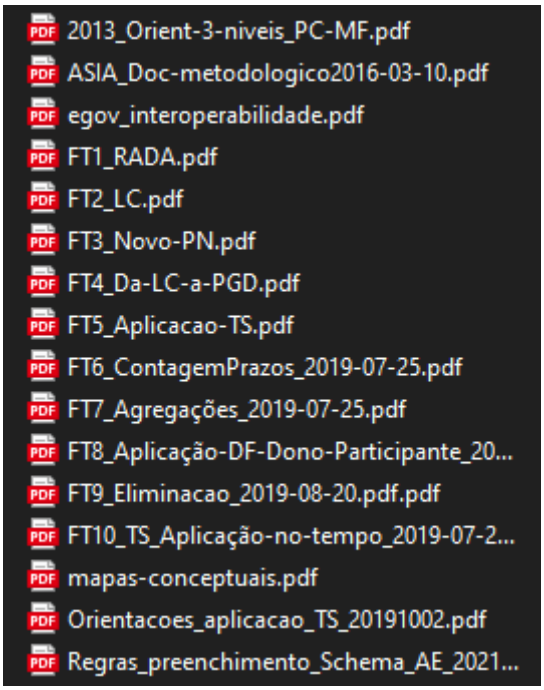


Figura 9: Conteúdo da diretoria ficheiros

Rada Old, PGD, PGDLC

Nestes casos, a aplicação começa por fazer uma chamada à API do CLAV de modo a obter a lista completa do tipo de informação em causa. Esta lista é depois percorrida e para cada elemento é feito um outro pedido à API do CLAV, de forma a obter a informação completa desse mesmo elemento. Esta informação é depois escrita num ficheiro JSON criado. que irá conter os dados relativos a esse tal elemento. É, também, calculado o *checksum* de cada ficheiro e escrito no manifesto. Todos estes ficheiros são, como habitual, guardados numa diretoria dentro da diretoria data.

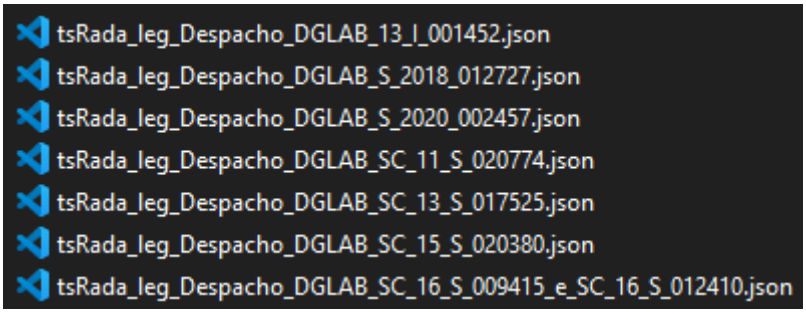


Figura 10: Exemplo dos ficheiros Rada Old

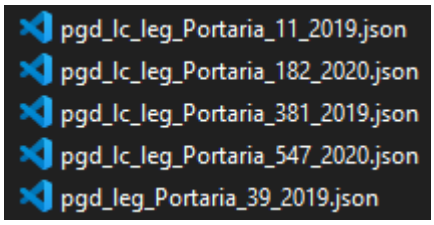


Figura 11: Exemplo dos ficheiros PGDLC

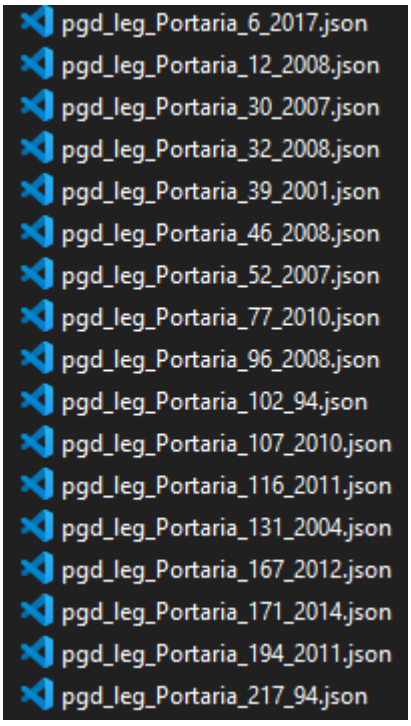


Figura 12: Exemplo dos ficheiros PGD

Classes

Para o caso das classes, é feito um pedido à API do CLAV para obter a lista das classes, com estrutura lista e com a informação completa. Depois, a aplicação comporta-se como para as informações do primeiro tópico, havendo apenas diferenças na parte das estatísticas. Neste caso, a aplicação conta o número de classes de nível 1,2,3 e 4 e não o número de objetos presentes na informação recebida pelo pedido.

4.4.6 *Finalização do pacote*

Estando toda a informação guardada, todos os ficheiros de meta-informação e estatística prontos e o manifesto escrito, o pacote é convertido para um zip, utilizando o módulo *AdmZip* do JavaScript, de modo a poder ser transferido por um utilizador. Este zip tem o mesmo nome do pacote e encontra-se também no *fileStore*.

4.5 Automatização de *backups* com N8N

De modo a poder automatizar a execução de *backups*, foi utilizado o serviço N8N. O N8N permite interagir com outras APIs e, entre outra funcionalidades, executar pedidos a APIs automáticos, num dado horário periódico. Neste caso, foi escolhido que seria executado um *backup*, todas as semanas às 00:00 de segunda-feira. De modo a implementar esta funcionalidade, foi utilizada a *interface* do serviço que pode ser acedida na porta 5678 do *localhost*. Aí, é possível criar os *workflows* e editá-los. Estes consistem em nodos, cada um com uma função, que executam os processos que lhe estão associados. Neste caso, foram utilizados dois nodos: um do tipo *cron*, que funciona como um *trigger* sempre que um período de tempo previamente escolhido passa e outro do tipo *http request*, que permite executar pedidos a APIs. Estes nódulos encontram-se conectados, ou seja, o pedido à API só é executado quando o nodo de *cron* lhe dá *trigger*.

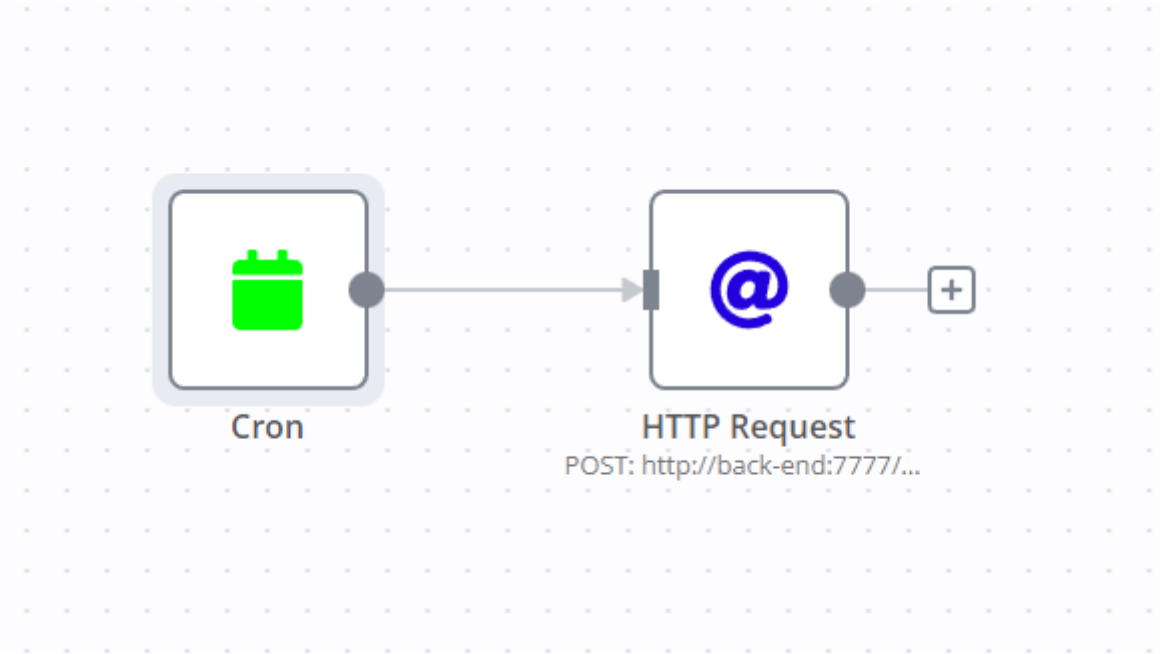


Figura 13: *Workflow* utilizado

Estando os nodos conectados, estes foram editados de modo a poderem executar de acordo com o modo em cima enunciado. Para editar um nodo basta clicar nele, sendo aberta a janela de edição. No caso do nodo de *cron*, a janela de edição permite escolher o período de tempo em que vai sendo ativado o *trigger* que faz com que o pedido à API seja executado. Foi portanto escolhido, de semana a semana, às 00:00 de cada segunda-feira.

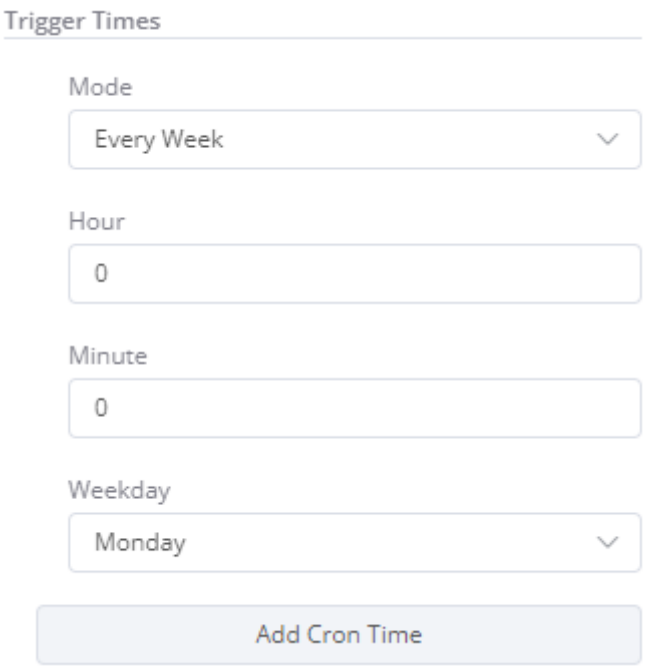


Figura 14: Edição do nodo cron

Quanto ao nodo *http request*, a janela de edição permite editar o url, tipo e *body* do pedido. Também permite, entre outras, uma opção *retry on fail* que faz com que o pedido seja executado de novo, passado um dado período de tempo, em caso de erro.

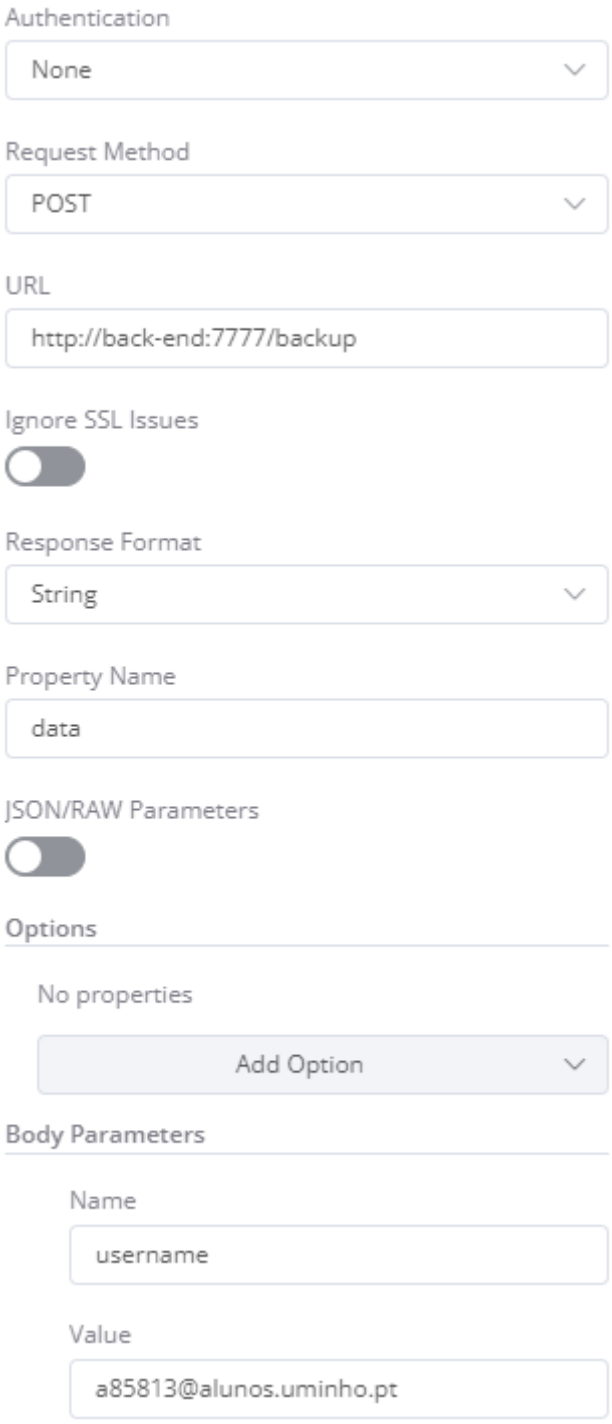


Figura 15: Edição do nodo http *request*

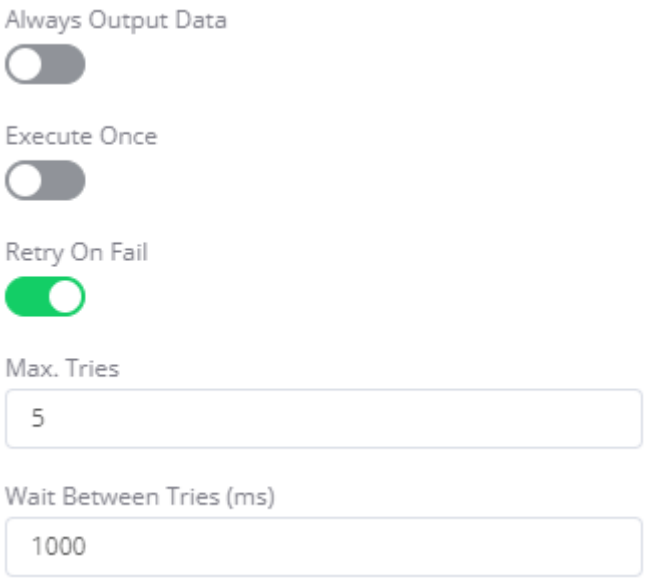


Figura 16: Edição do nodo http *request*

Por fim, basta ativar o *workflow*, premindo o botão *activate* no canto superior direito e o mesmo está pronto, executando de acordo com o que foi definido.

4.6 GESTÃO DE *backups*

Sempre que é finalizado um processo de *backup*, é também guardado numa base de dados em *MongoDB* (MongoDB), o nome, tamanho, criador, data de criação e ficheiros contidos no pacote. Isto tem como função permitir a um utilizador gerir os *backups* já criados e transferir ou apagar os pacotes. Para acomodar estas funcionalidades foi criada uma rota POST /bagit (4.2.2) que permite aceder à lista dos pacotes de *backup* existentes, mas requer um *username* e *password* de uma conta CLAV no body. Isto faz com que apenas utilizadores do CLAV possam aceder a esta informação. Foram também criadas uma rota de *delete* /bagit/:nome (4.2.4), que também requer autenticação, e uma rota de *download* GET /bagit/*download]/:nome* (4.2.3).

4.7 RESUMO

Neste capítulo, é descrito o processo de execução de *backups*. Desde os tipos de autenticação à criação e estrutura dos pacotes que contêm a informação.

É também abordada a automatização dos *backups* através da ferramenta N8N e ainda o processo de gestão dos *backups* já executados.

5

IMPORTAÇÃO DE DADOS

A importação de informação é feita através da ingestão de um pacote de *backup*. Apenas a informação presente no pacote é reposta, de uma das seguintes formas:

- **Importação total**: Este caso é utilizado quando a base de dados do CLAV encontra-se totalmente vazia, repondo toda a informação que se encontra no pacote;
- **Adição**: Neste caso, apenas é importada a informação presente no pacote, mas não presente na base de dados do CLAV;
- ***Overwrite*** Neste modo, todos os tipos de informação que se encontram no pacote são apagados da base de dados do CLAV. Feito isto, é inserido na base de dados CLAV a informação em si, presente no pacote.

De modo a implementar esta funcionalidade, foi criada uma rota POST */import* (4.2.5). Esta rota contém uma *query string* que indica o *token* de autenticação no CLAV (no caso da importação não pode ser utilizada uma *apikey*) e uma de nome modo, que explicita o modo de importação que será utilizado. Este modo será um dos três acima enunciados. Como se trata de uma rota onde é feito o *upload* de um pacote, é também enviado um ficheiro zip quando é feito um pedido na mesma.

5.1 PROCESSAMENTO DE PACOTE RECEBIDO

A aplicação começa por colocar o ficheiro zip recebido na diretoria *uploads*. Depois, utilizando o módulo de JavaScript *AdmZip* extrai o conteúdo do *zip* e coloca o mesmo numa diretoria de nome *extract*, que se encontra dentro da diretoria *public*. Como todos os pacotes contêm uma diretoria de nome *data*, contendo várias diretorias, uma para cada tipo de informação, a aplicação utiliza o módulo de *File System* do JavaScript (fs) para fazer um *readdirSync* da mesma, que retorna uma lista com os nomes das diretorias nela presentes. Através deste processo, é possível saber quais serão os tipos de informação presentes no pacote e processá-los um a um.

Para cada elemento da lista, é feito um *readFileSync* (também utilizando fs), ou seja, a leitura do ficheiro JSON presente na diretoria. Este processo permite aceder ao conteúdo do JSON, de modo a poder processá-lo objeto a objeto.

5.2 CONVERSOR JSON PARA TURTLE

Algo a ter em conta, também, é o facto de o CLAV utilizar 2 bases de dados diferentes: MongoDB (MongoDB) e GraphDB (GraphDB). Da informação disponível para importação, apenas os utilizadores e os pedidos são guardados em MongoDB, sendo o resto armazenado em GraphDB. Para este útlimo caso, é então necessário convertar a informação presente no pacote que está em formato JSON para o formato Turtle (Turtle) (uma sintaxe do tipo RDF (RDF)), utilizado pelo GraphDB. O Turtle consiste na utilização de triplos para o armazenamento de informação. Cada triplo contém um predicado que conecta dois objetos ou atributos, criando assim relações.

```
clav:ent_AR clav:entDesignacao "Assembleia da Republica"
```

Listing 5.1: Exemplo de um triplo em Turtle

O triplo em cima, relaciona um objeto (ent_AR) com um atributo, que neste caso é a sua designação (Assembleia da República). De referir, que o MongoDB suporta todos os formatos de serialização do RDF. De modo a executar a conversão, foi criado um conversor em JavaScript. Este conversor contém várias funções, uma para cada tipo de informação, que fazem a conversão de JSON para Turtle. Cada uma destas funções recebe um *array* de objetos em JSON e, tendo em conta o tipo de informação, vai fazendo a conversão objeto a objeto, retornando depois uma *string* em formato Turtle.

```
function tipologiaToTtl(obj){
  var query = ``
  obj.forEach(element => {
    let res = `
    clav:`+ element.id +` rdf:type owl:NamedIndividual, clav:TipologiaEntidade ;
        clav:tipDesignacao "`+ element.designacao +`";
        clav:tipEstado "`+ element.estado +`";
        clav:tipSigla "`+ element.sigla +`";`

    query = query + '\n' + res + '.'
  })

  return query
}
```

Listing 5.2: Conversão das tipologias

No exemplo acima, onde é feita a conversão das tipologias, podemos observar que para cada objeto do *array*, é criada uma *multi line string* onde é construído o formato Turtle. Depois de processada a tal *multi line string*, é adicionada uma *string* global de nome *query*, que no fim é retornada como resultado final da conversão. Esta lógica é utilizada para todos os tipos de informação em que a conversão é necessária.

5.3 IMPORTAÇÃO TOTAL

Neste tipo de importação, a aplicação simplesmente tem de inserir a informação que leu dos ficheiros JSON no CLAV. Um a um, cada objeto JSON presente no ficheiro é inserido, utilizando as rotas POST de repor da API do CLAV. No caso da informação que é armazenada em GraphDB, é primeiro feita a conversão para Turtle e depois, por fim, a sua inserção no CLAV. Este processo é feito para todos os tipos de informação presentes na diretoria data do pacote.

```
async function total(obj, token){
  for(const element of obj) {
    let query = converter.tipologiaToTtl([element])

    await axios({
      method: 'post',
      url: "http://clav-api.di.uminho.pt/v2/tipologias/repor?token=" + token,
      headers: {"User-Agent": "PostmanRuntime/7.26.8"},
      data: {
        query: query
      }
    })
    .then(() => {
      console.log("triplo " + element.id + " inserido")
    })
    .catch(e => console.log("ERRO na insercao dos triplos: " + element.id))
  }
}
```

Listing 5.3: Importação total das tipologias

5.4 ADIÇÃO

Neste caso, a aplicação começa por verificar se cada objeto do ficheiro JSON já se encontra no CLAV, utilizando as rotas GET /:id de cada tipo de informação. Caso um objeto não se encontre no CLAV, é prontamente inserido utilizando a respetiva rota de POST. Como

sempre, em tipos de informação que requeiram conversão, a mesma é feita antes da inserção. Este processo é feito para todos os tipos de informação presentes na diretoria *data* do pacote.

```
async function add(obj, token){
  for(const element of obj){
    await axios.get('http://clav-api.di.uminho.pt/v2/tipologias/' + element.id +
    '?token=' + token)
      .then(dados => {
        console.log(element.id + " existe")
      })
      .catch( async e => {
        console.log("Tipologia nao existe: " + element.id)
        let query = converter.tipologiaToTtl([element])
        await axios({
          method: 'post',
          url: "http://clav-api.di.uminho.pt/v2/tipologias/repor?token=" + token,
          headers: {"User-Agent": "PostmanRuntime/7.26.8"},
          data: {
            query: query
          }
        })
        .then(() => {
          console.log("triplo " + element.id + " inserido")
        })
        .catch(e => console.log("ERRO na insercao dos triplos: " + element.id))
    })
  }
}
}
```

Listing 5.4: Adição das tipologias

5.5 *overwrite*

Neste tipo de importação, a aplicação começa por, para cada tipo de informação presente na diretoria *data* do pacote, apagar toda essa informação do CLAV utilizando as rotas DELETE. Feito isto, é inserida a informação presente no pacote, utilizando o mesmo método da importação total.

```
async function overwrite(obj, token){
  await axios.get('http://clav-api.di.uminho.pt/v2/tipologias?token=' + token)
  .then(async dados => {
    for(const t of dados.data){
      await axios.delete('http://clav-api.di.uminho.pt/v2/tipologias/' + t.id  +
    '?token=' + token)
        .then(() => {
```

```
7             console.log(t.id + ' eliminada')
            })
9           .catch(err => {
              console.log(err.message)
11          })
      }
13    await total(obj, token)
    })
15  .catch(e => {
      console.log(e.message)
17  })
}
```

Listing 5.5: *Overwrite* das tipologias

5.6 RESUMO

Nesta capítulo, é abordado todo o processo de importação dos dados. Isto incluí o processamento dos pacotes recebidos e o tratamento dos dados neles presentes. São também explicados os três tipos possíveis de importação de dados: importação total, adição e *overwrite*.

6

INTERFACE WEB

A interface web da aplicação foi implementada utilizando a *framework* VueJS.

6.1 AUTENTICAÇÃO

Ao ser carregada, a aplicação começa por mostrar a sua página inicial.

Figura 17: Página Inicial

Esta página permite aceder à página de autenticação, clicando em *login* no canto superior direito da barra de navegação.

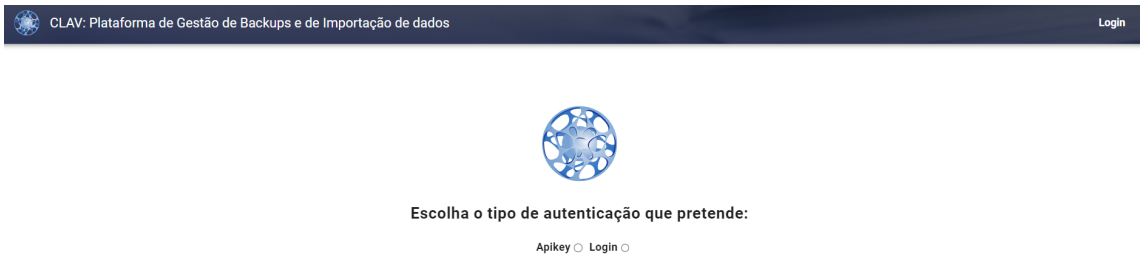


Figura 18: Página de autenticação

Nesta página, o utilizador deve escolher o método de autenticação que pretende utilizar: *login* ou *apikey*. Feito isto, surgem campos para que o utilizador possa escrever as suas credenciais.

Figura 19: Autenticação com Apikey

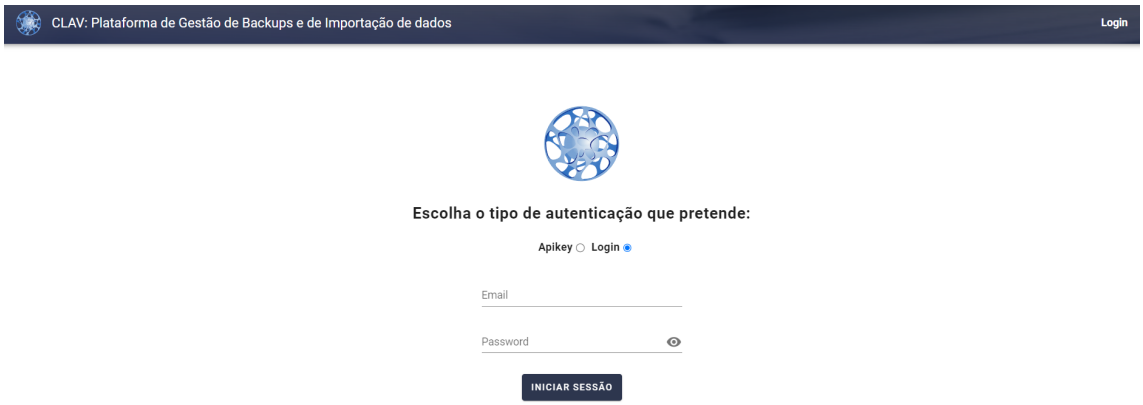


Figura 20: Autenticação com Login

Após a autenticação, o utilizador pode agora aceder às operações da aplicação: Execução e gestão *Backup* e Importação de dados, caso a autenticação seja feita com credenciais, ou apenas *backup* se for utilizada uma *apikey*.

Figura 21: Interface após autenticação

Sempre que é feita a autenticação, a aplicação guarda em *cookies*, o *token* gerado, o *email* e a *password* do utilizador, ou apenas *apikey* (dependendo do tipo de autenticação utilizado), de modo a poder utilizá-los nos pedidos que executa à API sem necessitar de autenticações constantes da parte do utilizador. Sempre que é feito um *logout*, estas *cookies* são apagadas. De modo a implementar esta função, foi utilizado o módulo *Vue cookies* do VueJS, que permite criar, guardar, aceder e eliminar *cookies*.

6.2 *backups*

Como já foi referido anteriormente, esta operação é permitida para os dois tipos de autenticação. No entanto, um utilizador com *apikey*, não pode executar *backup* de todos os tipos de informação, uma vez que não tem acesso a toda ela.

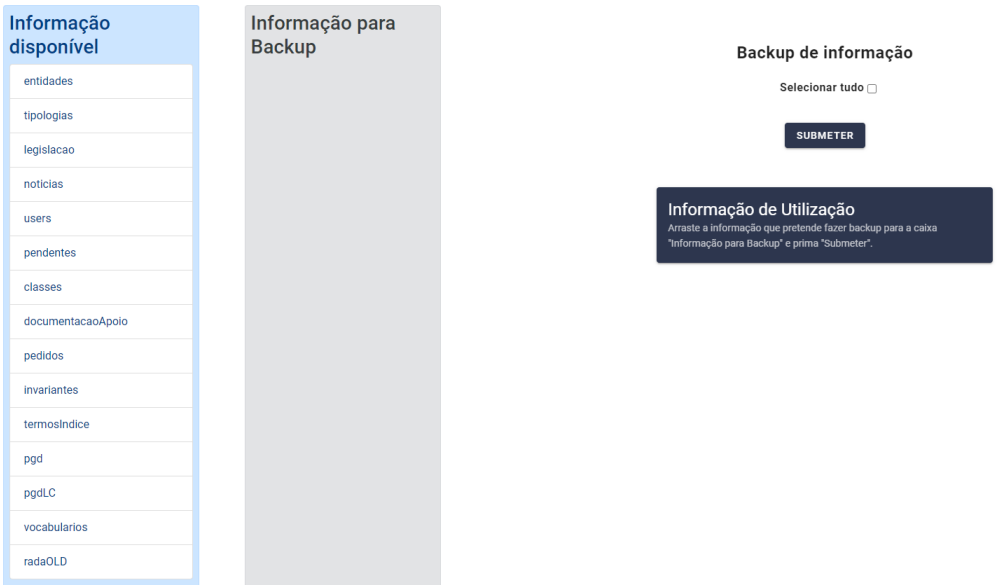


Figura 22: Interface de execução de *Backup* com autenticação por credenciais

Figura 23: Interface de execução de *Backup* com autenticação por *apikey*

Para executar o *backup*, o utilizador deve arrastar os tipos de informação que pretende recuperar da caixa "Informação disponível"para a caixa "Informação para *Backup*"e depois premir o botão de submissão. Caso um utilizador pretenda executar *backup* de toda a informação, pode premir em "Selecionar tudo", que coloca todos os tipos de informação na caixa "Informação para *Backup*". Feito isto, o *backup* é executado e quando estiver finalizado, o utilizador pode transferi-lo para a sua máquina.

6.3 GESTÃO DE *backups*

Esta funcionalidade só pode ser acedida por utilizadores com uma autenticação através de credencais. Nesta interface, o utilizador pode gerir os *backups* já executados, podendo saber o seu nome, criador, tamanho, data e hora de criação e ainda os ficheiros contidos no pacote. É, também, possível transferir ou eliminar qualquer um deles, bem como visualizar a lista dos *backups* em modo tabela ou em modo mosaico.

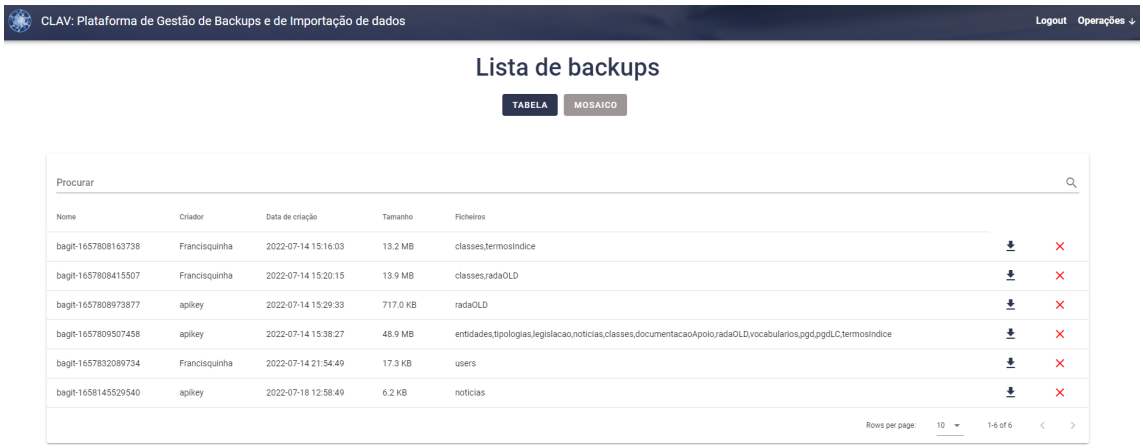


Figura 24: Interface da gestão de *backups* em modo tabela

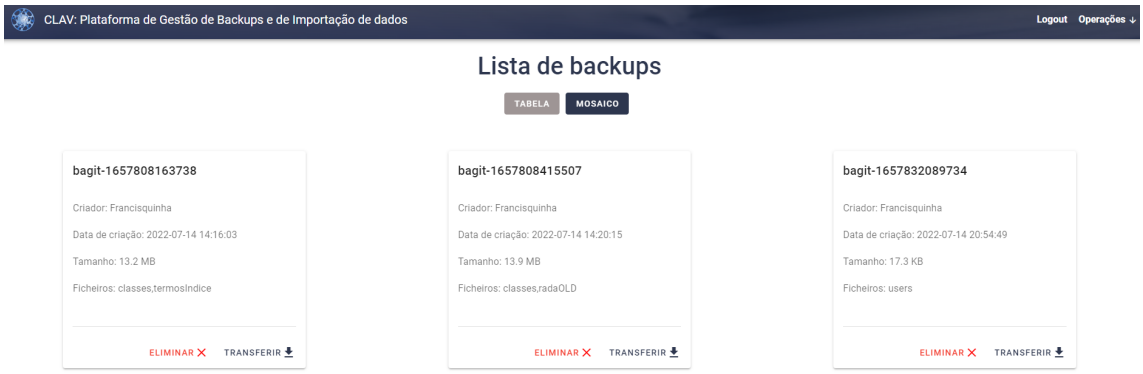


Figura 25: Interface da gestão de *backups* em modo mosaico

6.4 IMPORTAÇÃO

Esta funcionalidade, também apenas acessível a utilizadores autenticados com credencais, permite importar a informação presente num pacote de *backup* para o CLAV. A importação pode ser feita utilizando um dos três modos já enunciados no capítulo 5. Na interface, o utilizador deve selecionar o pacote zip que pretende importar, selecionar o tipo de informação e clicar em "Submeter"para inicializar a importação. Antes da importação começar, existe ainda um modal de confirmação, que explica o tipo de importação escolhido e pergunta ao utilizador se é o que pretende. Após a confirmação, a importação inicia.

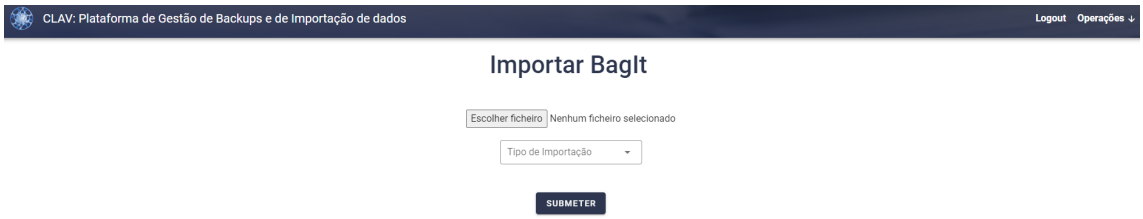


Figura 26: Interface de importação

6.5 RESUMO

Neste capítulo, é abordada a interface da aplicação *web* e descritas todas as suas funcionalidades e o seu modo de utilização.

7

DEPLOYMENT

De modo a permitir o *deployment* da aplicação, assim como a sua instalação, foi utilizado o *Docker* (Docker). Foram criados quatro *containers*, um para cada serviço, e depois utilizado um *docker-compose* (Docker Compose) para orquestrar todos os *containers*. Neste capítulo, são explicados todos os processos acima enunciados.

7.1 CONTAINER SERVIDOR BACKEND

A configuração do *container* do *backend* encontra-se no ficheiro *docker-compose*.

```
    back-end:
2       container_name: back-end
        restart: always
4       build:
            context: ./backend
6           dockerfile: ./Dockerfile
        links:
8           - mongo
        networks:
10          default:
              aliases:
12              - front-end
        volumes:
14          - ./backend/public/fileStore:/api/public/fileStore
```

Listing 7.1: Configuração do *container* do *backend*

O *container* é chamado de *back-end* e contém a opção de *restart*. Está conectado ao *container* do MongoDB, de modo a poder utilizar a base de dados e partilhar a rede com o *container* do *frontend* para poderem ser recebidas as chamadas à API. Em termos de volumes, é apenas partilhada a diretoria *filestore*, onde são guardados os pacotes de *backup*, de modo a permitir consistência em caso de falha do servidor. Quanto ao *build*, é utilizado um *Dockerfile* que executa esses processos.

```
FROM node:15

WORKDIR /api

COPY package.json /api
COPY package-lock.json /api
RUN npm install
COPY . /api

EXPOSE 7777

CMD [ "npm", "start" ]
```

Listing 7.2: *Dockerfile Backend*

O *Dockerfile* começa por importar a imagem base do NodeJS (neste caso 15). Posteriormente, cria dentro da imagem uma diretoria de nome api, copiando, de seguida, os ficheiros *package.json* e *package-lock.json* para essa mesma diretoria. De seguida, instalam-se as dependências com o comando *npm install*, sendo por fim todos os ficheiros copiados para a diretoria api. Por fim, é exposta a porta 7777, onde irá correr o servidor, e iniciado o mesmo com o comando *npm start*.

7.2 *container* SERVIDOR *frontend*

A configuração do *container* do *backend* encontra-se no ficheiro *docker-compose*.

```
    front-end:
        container_name: front-end
        restart: always
        build:
            context: ./frontend
            dockerfile: ./Dockerfile
        ports:
            - "7783:80"
        links:
            - back-end
        networks:
            default:
              aliases:
                - back-end
```

Listing 7.3: Configuração do *container* do *frontend*

O *container* é chamado de *front-end* e contém a opção de *restart*. Está conectado e partilha a rede com o *container* do *backend*, de modo a poder fazer os pedidos à API da aplicação. Quanto ao *build*, é utilizado um *Dockerfile* que executa esses processos.

```
FROM node:lts-alpine as build-stage
WORKDIR /frontend
COPY package*.json /frontend/
RUN npm install
COPY . /frontend/
RUN npm run build


FROM nginx:stable-alpine as production-stage
RUN rm /etc/nginx/nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=build-stage /frontend/dist /usr/share/nginx/html
COPY ./nginx.conf /etc/nginx/
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Listing 7.4: *Dockerfile Frontend*

O *Dockerfile* começa por importar a imagem base do NodeJS. Posteriormente, cria dentro da imagem uma diretoria de nome *frontend*, copiando, de seguida, os ficheiros *package.json* e *package-lock.json* para essa mesma diretoria. De seguida, instalam-se as dependências com o comando *npm install*, sendo por fim todos os ficheiros copiados para a diretoria *frontend*. Por fim, são feitas as configurações necessárias para o funcionamento do NGINX, ficando este hospedado na porta 80.

7.3 CONTAINER MONGODB

A configuração deste *container* encontra-se no ficheiro *docker-compose*.

```
    mongo:
        container_name: mongo
        restart: always
        environment:
            MONGO_INITDB_DATABASE: Backup
        image: mongo
        volumes:
            - ./mongo-volume:/data/db
```

Listing 7.5: Configuração do *container* do MongoDB

O *container* é nomeado de mongo, tendo a opção de *restart* ativada. É, também, indicado o nome da base de dados a ser utilizada (*Backups*) e os volumes partilhados que serão

as diretorias onde o MongoDB guarda os seus documentos, de modo a que estes estejam preservados em caso de falha do servidor.

7.4 CONTAINER N8N

A configuração deste *container* encontra-se no ficheiro *docker-compose*.

```
    n8n:
2       image: n8nio/n8n
        container_name: n8n
4       restart: always
        ports:
6       - 5678:5678
        links:
8           - back-end
        networks:
10          default:
              aliases:
12              - back-end
        volumes:
14          - ./n8n:/home/node/.n8n
```

Listing 7.6: Configuração do container do N8N

O *container* é chamado n8n, com a opção de *restart* ativa. Está hospedado na porta 5678, sendo nesta possível aceder à *interface web* de edição de *workflows*. Como o n8n comunica com o *backend* para fazer os pedidos de *backups*, estes estão conectados e partilham a mesma rede. Quanto aos volumes, é partilhada a diretoria onde são guardados os *workflows* e toda a sua informação, de modo a que esta não seja perdida em caso de falha do servidor.

7.5 INSTALAÇÃO

De modo a instalar a aplicação e poder executar a mesma, basta abrir um termninal dentro da diretoria onde se encontra a aplicação. Esta diretoria, onde também se encontra o ficheiro *docker-compose*, é responsável por orquestrar todos os *containers* já referidos. Depois, basta escrever o comando *docker-compose up* e aguardar que todas as instalações sejam concluídas. Estando este processo terminado, a aplicação está instalada e pronta a executar na máquina.

7.6 RESUMO

Neste capítulo é descrito o processo de *deployment* e instalação da aplicação através da utilização do Docker.

8

CONCLUSÃO

A presente dissertação assentou em dois objetivos principais: a execução de *backups* de informação presente no CLAV e a importação de informação através dos pacotes gerados em *backups*.

A execução de *backups* foi alcançada, sendo gerados pacotes zip em formato *BagIt*, contendo a informação e ainda meta-informação relativa à mesma. Foi, também, implementada uma automatização de *backups*, sendo executado um todas as segundas-feiras às oh, com toda a informação disponível. Os pacotes gerados podem ser geridos pelos utilizadores, sendo possível transferir ou apagar pacotes já gerados e ainda ter acesso a dados dos mesmos, tais como: tamanho, criador, ficheiros presentes e data de criação.

Nas questões de investigação formuladas, o formato escolhido para o pacote de *backup* gerado foi o *BagIt*, uma vez que se trata de uma alternativa mais simples em termos de implementação, mais percetível e de fácil utilização para o utilizador, em comparação com o *EARK-SIP*. Em relação à meta-informação a apresentar no pacote, foi decidido utilizar, para além da meta-informação pré definida do formato *BagIt*, o número de objetos de cada tipo de informação presente no pacote. Estes números encontram-se no ficheiro stats.html e podem ser úteis para comparação com outros pacotes e para conhecerr a quantidade de informação que possuí o pacote.

A importação de dados era em si uma questão de investigação, uma vez que era necessário saber como este processo poderia ser feito, ou seja, como poderia ser importada a informação presente num pacote para a plataforma CLAV. Este processo foi realizado com o módulo *AdmZip*, que permite extrair o conteúdo de ficheiros zip. A partir daqui, a aplicação acedia às diretorias presentes na diretoria *data* do pacote de modo a poder aceder à informação em si e por fim, importá-la para o CLAV através da API. Nesta questão foi também decidido que apenas utilizadores com autenticação por credenciais no CLAV poderiam executar esta operação, uma vez que, trata-se de algo com elevada importância e que acede diretamente aos dados da plataforma.

BIBLIOGRAFIA

Karin Bredenberg, Björn Skog, Anders Bo Nielsen, Kathrine Hougaard Edsen Johansen, Alex Thirifays, Sven Schlarb, and Andrew Wilson et al. Common Specification for Information Packages (Csip). *ERCIM News*, 2018.

Docker. Docker Documentation, 5 2022. URL https://docs.docker.com/. Acedido a 2022-05-15.

Docker Compose. Install Docker Compose, 5 2022. URL https://docs.docker.com/compose/install/. Acedido a 2022-05-15.

Rita Gago Filipa Carvalho, Helena Neves and Alexandra Lourenço. Aplicação de uma tabela de seleção. URL http://arquivos.dglab.gov.pt/wp-content/uploads/sites/16/2017/08/FT5_Aplicacao-TS.pdf. Acedido a 2021-11-12.

GraphDB. Ontotext. About GraphDB, 2 2022. URL http://graphdb.ontotext.com/documentation/free/about-graphdb.html. Acedido a 2022-02-12.

Brian Lavoie. The Open Archival Information System (OAIS) Reference Model: Introductory Guide (2nd Edition). *DPC Technology Watch Report*, 2014.

Library of Congress. Bagit: Transferring Content for Digital Preservation, 2009. URL https://www.loc.gov/item/webcast-4682/ Acedido a 2021-09-29.

Alexandra Lourenço, José Carlos Ramalho, Madalena Ribeiro, Pedro Penteado, and Rita Gago. Plataforma CLAV: garantindo a interoperabilidade semântica e preparando o acesso continuado à informação. *13.º Encontro Nacional de Arquivos Municipais, BAD, Cascais*, 2019.

José Carlos Lima Martins. CLAV: API de dados e Autenticação. Master's thesis, Universidade do Minho, 2020.

MongoDB. MongoDB Documentation, 10 2021. URL https://www.mongodb.com/docs/. Acedido a 2021-10-22.

RDF. Resource Description Framework (RDF): Concepts and Abstract Syntax , 3 2022. URL https://www.w3.org/TR/rdf-concepts/. Acedido a 2022-03-14.

Turtle. RDF 1.1 Turtle, 3 2022. URL https://www.w3.org/TR/turtle/. Acedido a 2022-03-14.

Clara Viegas and Alexandra Lourenço. O que é a Lista Consolidada, 12 2016. URL https://arquivos.dglab.gov.pt/wp-content/uploads/sites/16/2017/08/FT2_LC.pdf. Acedido a 2021-11-12.