



Universidade do Minho

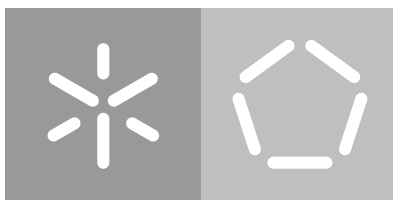
Escola de Engenharia

Departamento de Informática

José Carlos Lima Martins

CLAV:
API de dados e Autenticação

Outubro 2020



Universidade do Minho

Escola de Engenharia

Departamento de Informática

José Carlos Lima Martins

**CLAV:
API de dados e Autenticação**

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Trabalho realizado sob a orientação do Professor
José Carlos Leite Ramalho

Outubro 2020

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho



Atribuição

CC BY

<http://creativecommons.org/licenses/by/4.0/>

AGRADECIMENTOS

Quero agradecer ao meu orientador José Carlos Ramalho por todo o apoio e orientação durante esta etapa e pela oportunidade de trabalhar num projeto tão exigente, importante e gratificante como a [CLAV](#). Adicionalmente quero agradecer aos meus amigos e colegas Miguel Quaresma e João Vieira pelo apoio e orientação nos últimos 5 anos e ao meu amigo de infância Fábio Lopes pelo constante apoio. Quero também realizar um agradecimento especial ao meu pai e à minha mãe por todos os sacrifícios que fizeram e fazem para que tenha a melhor vida possível e pelo apoio durante esta etapa da minha vida. Finalmente, não posso deixar de agradecer aos familiares e amigos mais próximos no qual o seu apoio foi importante.

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.

RESUMO

A [Administração Pública](#) portuguesa tem desmaterializado processos e tem promovido a adoção de sistemas de gestão documental eletrónica bem como a digitalização de documentos destinados a serem arquivados. Estas medidas pretendem atingir a otimização de processos, a modernização de procedimentos administrativos e a redução de papel.

Com o propósito de atingir estes objetivos e simplificar a gestão documental na [Administração Pública](#), a [Classificação e Avaliação da Informação Pública \(CLAV\)](#) nasce como uma das medidas. A [CLAV](#) tem como finalidade a classificação e a avaliação da informação pública por forma a auxiliar os sistemas de informação das entidades públicas alertando-as quando determinado documento deve ser arquivado ou eliminado. Para tal esta possui um referencial comum, a [Lista Consolidada](#), com as funções e processos de negócio das entidades públicas associadas a um catálogo de legislação e de organismos.

Nos últimos dois anos, a [CLAV](#) tem vindo a ser desenvolvida no departamento de informática da [Universidade do Minho](#) em estreita colaboração com a equipa de investigação da área na [Direção-Geral do Livro, dos Arquivos e das Bibliotecas](#).

À data de início deste trabalho, a [CLAV](#) era constituída por dois servidores de bases de dados que tinham como interlocutor o servidor da [API](#) de dados da [CLAV](#). Era com este servidor da [API](#) de dados que toda a interação com o exterior passava: acesso de aplicações de terceiras partes e acessos da interface cliente desenvolvida para a [CLAV](#).

Nesta dissertação, o grande objetivo era fazer evoluir a arquitetura aplicacional dando resposta a uma série de requisitos e tentando simplificar ao máximo o processo da sua manutenção futura.

Nesse sentido, especificou-se e implementou-se um serviço para a proteção da [API](#) de dados da [CLAV](#), especificou-se a documentação desta [API](#) de dados, definiram-se os formatos de exportação e implementaram-se os exportadores desta [API](#) por forma a permitir uma maior interoperabilidade dos dados, implementou-se a autenticação com a [Chave Móvel Digital](#) recorrendo ao [Autenticação.gov](#), criaram-se os mecanismos necessários à migração de [HTTP](#) para [HTTPS](#) e, por fim, adicionou-se uma [API Gateway](#) na [CLAV](#) por forma a simplificar o funcionamento e gestão da plataforma.

Todos estes desenvolvimentos estão em produção e podem ser observados acedendo ao sítio Web oficial da [CLAV](#): <https://clav.dglab.gov.pt>

Keywords: API Gateway, Autenticação, Autenticação.gov, CLAV, Swagger

ABSTRACT

The portuguese public administration has dematerialized processes and promoted the adoption of electronic document management systems as well as the scanning of documents intended to be archived. This measures aim to optimize and modernize administrative procedures and reduce paper usage.

In order to achieve these objectives and simplify the document management in public administration, [CLAV](#) was born as one of the measures. [CLAV](#)'s main purpose is the classification and evaluation of the public information in order to help the information systems of public entities, alerting them when certain documents must be filed or deleted. To this end, a common reference, called the *consolidated list* ([Lista Consolidada](#)), is used, with the business functions and processes of public entities associated with a catalogue of legislation and entities.

In the last two years, [CLAV](#) have been develop in computing department of [UM](#) in strict collaboration with the area investigation team at [Direção-Geral do Livro, dos Arquivos e das Bibliotecas](#).

At start date of this work, [CLAV](#) was constituted by two database servers that had as interlocutor the data [API](#) server of [CLAV](#). Was with this data [API](#) server that all exterior interaction passed: access from third party applications and access from client interface developed for [CLAV](#).

In this dissertation, the big goal was make evolve the application architecture giving answer to a series of requirements and trying to simplify to maximum the process of it future maintenance.

In this sense, a protection service for data [API](#) of [CLAV](#) was specified and developed, the data [API](#) documentation was specified, the exportation formats were defined and the [API](#) exporters were developed in order to allow a bigger data interoperability, the authentication with [Chave Móvel Digital](#) using the [Autenticação.gov](#) was developed, the necessary mechanisms of [HTTP](#) to [HTTPS](#) migration were created and, lastly, an [API](#) Gateway on [CLAV](#) was added in order to simplify the operation and management of the platform.

All these developments are in production and can be observed accessing to the official web page of [CLAV](#): <https://clav.dglab.gov.pt>

Keywords: [API Gateway](#), [Autenticação.gov](#), [Authentication](#), [CLAV](#), [Swagger](#)

CONTEÚDO

1	INTRODUÇÃO	1
1.1	Motivação	2
1.2	Objetivos	2
1.3	Questões de investigação	3
1.4	Estrutura da dissertação	3
2	CLAV: PONTO DE PARTIDA	5
2.1	Conceitos	5
2.1.1	Entidade	5
2.1.2	Tipologia	5
2.1.3	Legislação	6
2.1.4	Processo de negócio	6
2.1.5	Lista Consolidada	6
2.1.6	Classificação	7
2.1.7	Avaliação	7
2.1.8	Tabela de Seleção	8
2.1.9	Donos e participantes de um Processo de Negócio	8
2.1.10	Autos de Eliminação	8
2.2	O papel da CLAV	9
2.3	Iniciativas Semelhantes à CLAV	9
2.3.1	Iniciativa Finlandesa	9
2.3.2	Iniciativa da Universidade <i>Victoria</i> de Toronto, Canadá	10
2.3.3	Comparação com a CLAV	10
2.4	Estrutura da plataforma CLAV	10
2.5	Formas de autenticação	12
2.5.1	Registo	13
2.5.2	<i>Login</i>	14
2.6	Resumo	16
3	PROTEÇÃO DA API DE DADOS	17
3.1	Estado da Arte	17
3.1.1	JSON Web Token (JWT)	17
3.2	Solução	23

3.3	Implementação	28
3.3.1	Interface da CLAV	30
3.4	Resumo	32
4	AUTENTICAÇÃO ATRAVÉS DE CMD	33
4.1	Estado da Arte	33
4.1.1	Autenticação.gov	33
4.2	Solução	38
4.3	Implementação	40
4.4	Resumo	41
5	DOCUMENTAÇÃO DA API DA CLAV	42
5.1	Estado da Arte	42
5.1.1	Swagger	42
5.1.2	Especificação <i>OpenAPI</i>	43
5.1.3	<i>Swagger UI</i>	50
5.1.4	Produção da documentação da API da CLAV	51
5.2	Solução	55
5.3	Implementação	57
5.4	Resumo	58
6	EXPORTAÇÃO DE DADOS	59
6.1	Estado da Arte	59
6.1.1	XML	59
6.1.2	CSV	64
6.1.3	Ontologia	67
6.2	Solução	70
6.2.1	XML	70
6.2.2	CSV	71
6.2.3	Ontologia	75
6.2.4	Exportação na API de dados	76
6.3	Implementação	77
6.3.1	XML	78
6.3.2	CSV	79
6.3.3	Exportação na API de dados	81
6.3.4	Interface de Exportação	81
6.4	Resumo	82
7	MIGRAÇÃO DE HTTP PARA HTTPS	83
7.1	Estado da Arte	83

7.1.1	Let's Encrypt	83
7.2	Solução	86
7.3	Implementação	88
7.4	Resumo	91
8	API GATEWAY	92
8.1	Estado da Arte	92
8.1.1	API Gateway	92
8.2	Solução	99
8.2.1	Arquitetura	101
8.3	Implementação	102
8.3.1	Serviço de <i>Auth</i>	102
8.3.2	<i>Plugin external-auth</i>	104
8.3.3	Configuração <i>Kong</i>	105
8.3.4	Arquitetura	111
8.4	Resumo	112
9	DEPLOYMENT	113
9.1	Versão sem <i>Kong</i>	113
9.1.1	Primeira instalação	113
9.1.2	Atualizações	119
9.1.3	Backup	120
9.1.4	Migração	121
9.2	Versão com <i>Kong</i>	122
9.2.1	Primeira instalação	122
9.2.2	Atualizações	124
9.2.3	Backup	124
9.2.4	Migração	125
9.2.5	Forçar renovação do certificado HTTPS	125
9.3	Versão sem <i>Kong</i> vs com <i>Kong</i>	125
9.3.1	Configuração do HTTPS	126
9.3.2	Resumo	127
9.4	Resumo	128
10	CONCLUSÕES E TRABALHO FUTURO	129
10.1	Trabalho Futuro	130

LISTA DE FIGURAS

Figura 1	Estrutura da CLAV incluindo a interação de um utilizador com a mesma	11
Figura 2	Estrutura evoluída da CLAV	12
Figura 3	Fluxo do <i>login</i> de um utilizador através do Autenticação.gov antes da limitação do registo de utilizadores apenas por pessoal autorizado	15
Figura 4	Exemplo de representação compacta de JWT (quebra de linhas por forma a melhorar leitura)	18
Figura 5	Criação de um JWT	21
Figura 6	Criação de um JWS	21
Figura 7	Estratégia de proteção da API de dados	25
Figura 8	Fluxo de autenticação e posteriores pedidos de um utilizador	26
Figura 9	Fluxo de autenticação e posteriores pedidos de uma chave API	27
Figura 10	Fluxo de pedidos entre a CLAV e o Autenticação.gov de forma a autenticar um utilizador na CLAV . [2]	34
Figura 11	Autenticação.gov : Processo de autenticação com CMD	40
Figura 12	Utilizador autenticado com sucesso através de CMD	40
Figura 13	Mensagem de erro na CLAV caso não se encontre registado na CLAV mas tenha se autenticado com sucesso no Autenticação.gov	41
Figura 14	Mensagem de erro na CLAV após falha no Autenticação.gov	41
Figura 15	<i>Swagger UI</i> exemplo	50
Figura 16	Interface de exportação	81
Figura 17	Exemplo de validação do domínio pelo <i>Let's Encrypt</i> com sucesso	85
Figura 18	Arquitetura a desenvolver com HTTPS	88
Figura 19	Arquitetura a desenvolver com API Gateway	101
Figura 20	Arquitetura desenvolvida com API Gateway	111
Figura 21	Parte do email recebido pelo <i>GraphDB</i>	115
Figura 22	Classificação <i>SSL Labs</i> da API sem <i>Kong</i>	126
Figura 23	Classificação <i>SSL Labs</i> da API com <i>Kong</i>	127
Figura 24	Classificação <i>SSL Labs</i> da Interface	127

LISTA DE TABELAS

Tabela 1	Comparação entre especificações de documentação de APIs	49
Tabela 2	Comparação entre ferramentas de APIs	51
Tabela 3	Rotas com exportação, formatos de saída disponíveis para cada rota e valores a usar por forma a exportar nesse formato de saída	77
Tabela 4	Comparação entre <i>API Gateways</i> [34, 32, 33]	100
Tabela 5	Variáveis ambiente do ficheiro <i>.env</i> da API de dados sem <i>Kong</i>	116
Tabela 6	Variáveis ambiente do ficheiro <i>.env</i> da interface	117
Tabela 7	Variáveis ambiente do ficheiro <i>.env</i> da API de dados com <i>Kong</i>	123
Tabela 8	Resultados de performance para a API de dados	126

LISTA DE EXEMPLOS

3.1	<i>Header</i> usado para construir o JWT da figura 4	19
3.2	<i>Payload</i> usado para construir o JWT da figura 4	20
3.3	<i>Signature</i> usado para construir o JWT da figura 4	20
3.4	Verificação se um pedido com uma determinada Chave API pode ser efetuado	28
3.5	Verificação se um pedido com um determinado <i>token</i> de um utilizador registado pode ser efetuado	29
3.6	Extração do <i>token</i> da <i>query string</i>	29
3.7	Extração do <i>token</i> da <i>header Authorization</i>	30
3.8	Verificação se um utilizador registado tem permissões suficientes para aceder a uma determinada rota	30
4.1	Extensão Nível de Confiança no pedido enviado ao Autenticação.gov	39
4.2	Extensão Política de Apresentação no pedido enviado ao Autenticação.gov	39
5.1	Exemplo de indicação da versão da especificação <i>OpenAPI</i>	43
5.2	Exemplo de secção <i>info</i> indicando título, descrição e versão da API na especificação <i>OpenAPI</i>	44
5.3	Exemplo de secção <i>servers</i> indicando os <i>URLs</i> e a descrição de cada na especificação <i>OpenAPI</i>	44
5.4	Exemplo de secção <i>paths</i> indicando os detalhes de cada rota na especificação <i>OpenAPI</i>	44
5.5	Exemplo de secção <i>tags</i> definindo <i>tags</i> na especificação <i>OpenAPI</i>	45
5.6	Exemplo de uso de <i>tags</i> numa rota na especificação <i>OpenAPI</i>	45
5.7	Exemplo de adição de exemplos para XML e HTML na especificação <i>OpenAPI</i>	48
5.8	Exemplo de uso do <code>swagger-ui-express</code>	52
5.9	Exemplo de uso do <code>yaml-include</code> no documento de especificação <i>OpenAPI(index.yaml)</i>	53
5.10	Exemplo de estrutura dos ficheiros para gerar o documento de especificação <i>OpenAPI</i>	54
5.11	Documento de especificação <i>OpenAPI</i> gerado a partir do ficheiro <i>index.yaml</i> com o uso da <i>package</i> <code>yaml-include</code>	54
5.12	Excerto da estrutura modular da documentação	56
6.1	Pequeno exemplo em XML	59
6.2	Exemplo em JSON a converter	61
6.3	Resultado da conversão do exemplo 6.2 usando o conversor <code>xml-js</code>	62
6.4	Código para a construção em XML do exemplo 6.1 usando o <code>xmlbuilder</code>	62

6.5	Resultado da conversão do exemplo 6.2 usando o conversor <code>xmlbuilder</code>	63
6.6	Resultado da conversão do exemplo 6.2 usando o conversor <code>xml2js</code>	64
6.7	Pequeno exemplo em <code>CSV</code>	65
6.8	Resultado da conversão do exemplo 6.2 usando o conversor <code>papaparse</code>	65
6.9	Resultado da conversão do exemplo 6.2 usando o conversor <code>json2csv</code>	66
6.10	Outro exemplo em <code>JSON</code> a converter	66
6.11	Resultado da conversão do exemplo 6.10 usando o conversor <code>json2csv</code>	67
6.12	Resultado pretendido da conversão do exemplo 6.10	67
6.13	<code>JSON</code> exemplo a converter	78
6.14	<code>XML</code> resultante da conversão do <code>JSON</code> presente em 6.13	78
6.15	<code>CSV</code> resultante da conversão do <code>JSON</code> presente em 6.13	80
7.1	Redirecionamento de <code>HTTP</code> para <code>HTTPS</code> e validação do domínio na configuração <code>Nginx</code> .	89
7.2	Certificado na configuração <code>Nginx</code>	89
7.3	Recomendações de segurança na configuração <code>Nginx</code>	89
8.1	Configuração declarativa do <code>Kong</code> : <code>API</code> de dados	105
8.2	Configuração declarativa do <code>Kong</code> : <code>plugin cors</code>	106
8.3	Configuração declarativa do <code>Kong</code> : <code>plugin rate-limiting</code>	106
8.4	Configuração declarativa do <code>Kong</code> : <code>plugin proxy-cache</code>	106
8.5	Configuração declarativa do <code>Kong</code> : <code>plugin response-transformer</code>	107
8.6	Configuração declarativa do <code>Kong</code> : Rota da documentação	108
8.7	Configuração declarativa do <code>Kong</code> : <code>plugin acme</code>	108
8.8	Configuração declarativa do <code>Kong</code> : Serviço para a geração de certificados <code>TLS</code>	109
8.9	Configuração declarativa do <code>Kong</code> : Serviço para a geração de certificados <code>TLS</code>	109
8.10	Configurações do <code>Nginx</code> no ficheiro de configuração <code>.conf</code>	110
9.1	Instalar <code>docker</code> e <code>docker-compose</code>	113
9.2	Backup dos volumes do <code>docker</code>	120
9.3	Restauro dos volumes do <code>docker</code>	121

LISTA DE ACRÓNIMOS

- ACME** Automatic Certificate Management Environment 84, 85, 87
- AE** Auto de Eliminação 8, 9
- AEAD** Authenticated Encryption with Additional Data 23
- AMA** Agência para a Modernização Administrativa 36
- AP** Administração Pública iv, 1, 6, 7, 33
- API** Application Programming Interface iv, v, ix–xii, 2, 3, 11–14, 16, 17, 23–32, 42–44, 48–51, 53, 55, 57, 59, 69, 76, 77, 81, 83, 86–109, 111, 112, 114–117, 119, 120, 122, 123, 126–131
- AWS** Amazon Web Services 95
- BD** Base de Dados 69, 75–77, 117, 118, 123
- CA** Certificate Authority 83, 86, 91
- CAA** Certificate Authority Authorization 119, 127
- CC** Cartão de Cidadão 12, 14, 33–39, 129
- CLAV** Classificação e Avaliação da Informação Pública ii, iv, v, ix, 1–3, 5, 8–13, 15, 16, 18, 22, 23, 30–38, 40–43, 57, 76, 77, 81, 83, 86, 91, 92, 94, 97, 100–103, 111, 112, 114, 116, 120, 123, 129
- CLI** Command-Line Interface 95, 96, 99, 100
- CMD** Chave Móvel Digital iv, v, ix, 3, 33, 34, 36–41, 129
- CORS** Cross-Origin Resource Sharing 11, 95–100, 106
- CPU** Central Processing Unit 98, 125
- CSP** Content Security Policy 86
- CSS** Cascading Style Sheets 11, 42
- CSV** Comma Separated Values xii, 3, 59, 64, 65, 71–75, 77, 79–82, 129–131
- DDoS** Distributed Denial of Service 94, 99
- DF** Destino Final 6–8
- DGLAB** Direção-Geral do Livro, dos Arquivos e das Bibliotecas iv, v, 1, 5, 6, 8, 9, 14
- DH** Diffie-Hellman 87, 88
- DNS** Domain Name System 84, 119, 127
- DOM** Document Object Model 63
- DV** Domain Validation 83, 84
- ERMS** Electronic Records Management System 9

- GUI** Graphical User Interface 95, 96, 98–100
- HMAC** Hash-based Message Authentication Code 17, 19, 22
- HSTS** HTTP Strict Transport Security 86, 119
- HTML** Hypertext Markup Language xi, 11, 42, 48
- HTTP** Hypertext Transfer Protocol iv, v, xii, 2, 3, 24, 31, 32, 47, 48, 54, 56, 69, 83, 84, 86, 89, 91, 94, 97, 102–106, 109–111, 114, 116, 117, 123, 129
- HTTPS** Hypertext Transfer Protocol Secure iv, v, viii, ix, xii, 2, 3, 36, 83, 86–89, 91, 95–100, 105, 107–111, 114, 116–119, 123, 125–127, 129
- iAP** Interoperabilidade na Administração Pública 3, 129
- IETF** Internet Engineering Task Force 23
- IP** Internet Protocol 106, 125
- JOSE** JSON Object Signing and Encryption 18, 19
- JSON** JavaScript Object Notation xi, xii, 3, 17, 18, 22, 43, 49, 50, 53, 59, 61–63, 65–67, 69, 70, 75, 77–82, 94, 96, 97, 105, 106, 131
- JSON-LD** JavaScript Object Notation for Linked Data 59, 69, 76, 77
- JWE** JSON Web Encryption 17, 18, 23
- JWS** JSON Web Signature ix, 17, 18, 20, 21, 23
- JWT** JSON Web Token ix, xi, 14–26, 32, 38, 95, 96, 98–100, 102
- LC** Lista Consolidada iv, v, 1, 2, 6–10, 12, 59, 130
- MIME** Multipurpose Internet Mail Extensions 69
- N3** Notation3 69
- NIC** Número de Identificação Civil 13, 34, 38, 40, 129
- NPM** Node Package Manager 59, 61, 65, 82
- NSA** National Security Agency 19
- OAS** OpenAPI Specification 42
- OASIS** Organization for the Advancement of Structured Information Standards 35
- OCSP** Online Certificate Status Protocol 35, 127
- OS** Operating System 109, 113
- OWL** Web Ontology Language 68, 69
- PCA** Prazo de Conservação Administrativa 6–8
- PDF** Portable Document Format 11
- PIN** Personal Identification Number 33–36, 38, 40

- PKI** Public Key Infrastructure 33, 35
- PN** Processo de Negócio 6, 8
- POSIX** Portable Operating System Interface 20
- RADA** Relatórios de Avaliação de Documentação Acumulada 10
- RAM** Random-Access Memory 125
- RAML** RESTful API Modeling Language 49, 51
- RDF** Resource Description Framework 3, 59, 68, 69, 76, 77, 82, 129
- REST** Representational State Transfer 2, 42, 43, 50, 51, 69, 96–98
- RIF** Rule Interchange Format 69
- RSA** Rivest–Shamir–Adleman 18, 19, 35
- SAIL** Storage And Inference Layer 69
- SAML** Security Assertion Markup Language 22, 35–37
- SDK** Software Development Kit 43
- SHA-2** Secure Hash Algorithm 2 19
- SKOS** Simple Knowledge Organization System 68, 69
- SMS** Short Message Service 33, 34, 40
- SNI** Server Name Indication 127
- SOAP** Simple Object Access Protocol 69
- SPARQL** SPARQL Protocol and RDF Query Language 68, 69, 94
- SQL** Structured Query Language 69, 94
- SSL** Secure Sockets Layer 36, 83, 90
- SSO** Single Sign On 18, 33, 35
- SWT** Simple Web Token 22
- TAR** Tape Archive 121
- TLS** Transport Layer Security xii, 83, 94, 109, 110
- TS** Tabela de Seleção 8, 10, 130
- Turtle** Terse RDF Triple Language 59, 69, 76, 77
- UI** User Interface ix, 42, 43, 50–53, 55
- UM** Universidade do Minho iv, v, 1
- URI** Uniform Resource Identifier 84
- URL** Uniform Resource Locator 36, 92, 115–117
- W3C** World Wide Web Consortium 68, 69
- WWW** World Wide Web 68

XML Extensible Markup Language xi, xii, 3, 22, 35, 48, 59–63, 69, 70, 76–82, 94, 129

XSLT Extensible Stylesheet Language Transformations 94

YAML YAML Ain't Markup Language 42, 43, 49, 50, 53, 55, 95, 96, 105

INTRODUÇÃO

Vemos atualmente a mudança de paradigma em várias organizações e governos em relação a políticas e estratégias para a disponibilização de dados abertos nos domínios das ciências e da [Administração Pública \(AP\)](#). Quanto à [Administração Pública](#) portuguesa têm sido promovidas políticas para a sua transformação digital com o objetivo de otimização de processos, a modernização de procedimentos administrativos e a redução de papel. De certa forma a agilização de procedimentos da [Administração Pública](#) portuguesa. [31]

De forma a alcançar estes objetivos a [Administração Pública](#) tem desmaterializado processos e tem promovido a adoção de sistemas de gestão documental eletrónica bem como da digitalização de documentos destinados a serem arquivados. [31]

Por forma a continuar esta transformação da [AP](#) a [Direção-Geral do Livro, dos Arquivos e das Bibliotecas \(DGLAB\)](#) apresentou a iniciativa da [Lista Consolidada \(LC\)](#) para a classificação e avaliação da informação pública. A [LC](#) serve de referencial para a construção normalizada dos planos de classificação e tabelas de seleção das entidades que executam funções do Estado. [31]

Neste contexto, nasce o projeto [Classificação e Avaliação da Informação Pública \(CLAV\)](#) sendo um dos seus objetivos primordiais a operacionalização da utilização da [LC](#), numa colaboração entre a [DGLAB](#) e a [Universidade do Minho \(UM\)](#) e financiado pelo [SIMPLEX¹](#). [31]

A plataforma [CLAV](#) disponibiliza em formato aberto uma ontologia com as funções e processos de negócio das entidades que exercem funções públicas (ou seja a [LC](#)) associadas a um catálogo de legislação e de organismos. Desta forma, a [CLAV](#) viabiliza a desmaterialização dos procedimentos associados à elaboração de tabelas de seleção tendo como base a [LC](#) e ao controlo de eliminação e arquivamento da informação pública através da integração das tabelas de seleção nos sistemas de informação das entidades públicas alertando-as quando determinado documento deve ser arquivado ou eliminado. Esta integração promove também a interoperabilidade através da utilização de uma linguagem comum (a [LC](#)) usada no registo, na classificação e na avaliação da informação pública. [31]

¹Programa de Simplificação Administrativa e Legislativa

1.1 MOTIVAÇÃO

A continuação do desenvolvimento da [API](#) de dados da [CLAV](#) nesta dissertação, seguindo uma metodologia [REST](#)², permite a processos ou aplicações aceder aos dados sem a intervenção humana para além de suportar a plataforma [CLAV](#). Um dos objetivos da [API](#) de dados é permitir futuramente a criação de novas aplicações através desta. Como tal, é extremamente essencial que a [API](#) de dados da [CLAV](#) possua uma boa documentação ajudando futuros programadores ou utilizadores a utilizar a [API](#). Além disso, uma [API](#) sem uma boa documentação de como a usar é inútil. Advém daí a necessidade de nesta dissertação realizar a documentação da [API](#) de dados em *Swagger*.

Apesar de o projeto ter em mente a disponibilização aberta de informação pública é necessário controlar a adição, edição e eliminação da informação presente na [Lista Consolidada](#), bem como a informação de utilizadores, da legislação, das entidades, etc, mantendo-a consistente e correta. É, portanto, necessário controlar os acessos à [API](#) de dados com múltiplos níveis de acesso restringindo as operações que cada utilizador pode realizar consoante o seu nível. Desta forma garante-se que apenas pessoal autorizado pode realizar modificações aos dados.

Este controlo de acesso exige a existência de formas de autenticação. Como um cofre para o qual ninguém tem a chave não é útil pelo facto de que algo lá guardado ficará eternamente inacessível, também algo com controlo de acesso seria inútil caso não fosse possível ultrapassar esse controlo de alguma forma. Assim, uma das formas de autenticação usadas, [Autenticação.gov](#), criada pelo Estado português, permite a autenticação dos cidadãos portugueses nos vários serviços públicos [3] entre os quais, a Segurança Social, o Serviço Nacional de Saúde e a Autoridade Tributária Aduaneira. Sendo este um projeto do Governo Português, a autenticação na [CLAV](#) através do [Autenticação.gov](#) é um requisito.

Por forma a contrariar o aumento da complexidade da [API](#) de dados com a adição do controlo de acesso e da autenticação pretende-se investigar se a criação de uma [API Gateway](#) simplifica a comunicação entre interface/utilizadores e a [API](#) de dados.

Por fim, um dos requisitos para usar o [Autenticação.gov](#) em produção é a plataforma [CLAV](#) estar com [HTTPS](#). Além disso, o Centro Nacional de Cibersegurança avisou para a necessidade da plataforma [CLAV](#) estar com [HTTPS](#) aumentando a segurança da plataforma. Assim, é necessário nesta dissertação realizar a migração de [HTTP](#) para [HTTPS](#) da plataforma [CLAV](#).

1.2 OBJETIVOS

Resumidamente, os objetivos desta dissertação são:

- Documentação em *Swagger* da [API](#) de dados da [CLAV](#)

²Mais informação em https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

- Adição de formatos de exportação à *API* de dados da *CLAV* (para além do já presente *JSON*, adicionar *CSV*, *XML* e *RDF*)
- (Continuação da) Integração do Autenticação.gov na *CLAV* (adição da autenticação através de *Chave Móvel Digital*)
- Proteção da *API* de dados da *CLAV* com múltiplos níveis de acesso
- Migração da plataforma *CLAV* de *HTTP* para *HTTPS*
- Criação de uma arquitetura aplicacional mais eficiente em termos de funcionamento e gestão (que inclui uma *API Gateway*)
- Integração da *CLAV* no *iAP*

1.3 QUESTÕES DE INVESTIGAÇÃO

Nesta dissertação, durante o seu desenvolvimento, levantaram-se algumas questões de investigação.

Sendo um dos objetivos a documentação da *API* de dados, uma questão que surge desde logo é qual a melhor forma de documentar a *API*.

Por outro lado, com a investigação da criação de uma *API Gateway* na *CLAV* surgiu a necessidade de autonomizar o serviço de autenticação (microserviço independente da *API* de dados) por forma a respeitar os requisitos da autenticação e proteção da *API* de dados. Será que esta autonomização traz benefícios à aplicação? Além disso, com a maior modularização da *API* de dados em microserviços será que a *API Gateway* permite melhorar esta arquitetura?

As respostas a estas questões estarão presentes na conclusão desta dissertação.

1.4 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação começa no capítulo 2 por apresentar o ponto de partida desta dissertação quanto ao estado da *CLAV*.

De seguida, abordam-se os vários objetivos da dissertação, um por capítulo, a proteção da *API* de dados no capítulo 3, a adição da autenticação através de *CMD* no capítulo 4, a documentação da *API* da *CLAV* no capítulo 5, a adição de formatos exportação à *API* de dados no capítulo 6, a migração de *HTTP* para *HTTPS* no capítulo 7 e a criação de uma *API Gateway* no capítulo 8. Em cada um destes capítulos inicia-se com a apresentação do estado da arte e conseqüente solução e implementação da solução por forma a atingir o objetivo.

Após a apresentação destes capítulos no capítulo 9 apresenta-se de que forma pode ser feita a instalação das duas versões desenvolvidas durante esta tese, a versão sem a presença do *Kong* e a versão com *Kong*. É, além disso, realizada uma comparação das duas versões.

Finalmente, no capítulo 10 são apresentadas as conclusões finais e os trabalhos futuros a efetuar.

CLAV: PONTO DE PARTIDA

Quando esta dissertação teve início o projeto [CLAV](#) já tinha cerca de 2 anos de desenvolvimento. Assim nesta secção será apresentado o estado da arte da [CLAV](#) quando esta dissertação iniciou, aprofundando principalmente os pontos mais importantes sobre o tema desta dissertação. Além disso, serão apresentados os principais conceitos da [CLAV](#) e iniciativas semelhantes a esta.

2.1 CONCEITOS

Para facilitar a compreensão e a leitura desta dissertação, explicam-se nesta secção os principais conceitos da [CLAV](#).

2.1.1 *Entidade*

Na [CLAV](#) quando se faz referência a uma entidade esta é uma entidade pública. As entidades públicas intervêm nos processos de negócio podendo ter um de dois níveis de responsabilidade: uma entidade pode ser dona do processo ou participante no mesmo. Alguns exemplos de entidades públicas são hospitais públicos, escolas públicas, universidades públicas ou por exemplo, de forma mais específica, a [Direção Geral de Viação](#), [Instituto Nacional de Estatística](#), [Direção-Geral do Livro, dos Arquivos e das Bibliotecas](#), entre outros.

2.1.2 *Tipologia*

Uma tipologia corresponde a um agrupamento de entidades públicas e surge no contexto deste projeto para facilitar algumas operações de associação e relacionamento que envolvem um grande número de entidades. Alguns exemplos são [Agrupamentos de Centros de Saúde](#), [Autarquias Locais](#), [Forças Armadas](#), [Embaixadas](#), entre outros.

2.1.3 Legislação

A legislação, no contexto da CLAV, corresponde ao conjunto de documentos legislativos que regulam ou condicionam as atividades desenvolvidas nos processos de negócio da administração pública e enquadram os prazos de conservação administrativa (PCA) e o destino final (DF) destes processos de negócio. O PCA é o período de tempo, registado em anos, durante o qual a informação deve ser mantida para responder a necessidades de negócio, requisitos organizacionais, responsabilização e obrigações legais [16]. O DF é o destino final a dar à informação depois de cumprido o PCA. O DF pode ser de *Conservação* (a documentação do processo tem que ser preservada em arquivo definitivo ou histórico), de *Conservação Parcial* (a documentação tem que ser preservada por amostragem, apenas uma parte é preservada) ou de *Eliminação* (a documentação é fisicamente eliminada sendo registada essa ação de eliminação).

2.1.4 Processo de negócio

Um **Processo de Negócio (PN)** é uma sucessão ordenada de atividades interligadas, desempenhadas para atingir um resultado definido (produto ou serviço), no âmbito de uma função. [44]

Na LC o PN é representado por uma classe de 3º nível que pode ser subdividida em classes de 4º nível nos casos em que os vários documentos gerados ao longo das etapas do **Processo de Negócio** possuam diferentes PCA's e/ou diferentes DF's.

Um exemplo de PN é o processamento de matrículas ou inscrições no ensino ou em formação, ou seja, a realização ou renovação de matrícula em cursos ou inscrição em ações de formação. Este PN inicia com o pedido de acesso ou ingresso e termina com a entrega de comprovativo de matrícula ou inscrição. Pelo meio possui etapas como a verificação de dados de identificação e validação da existência dos requisitos necessários para efeito de matrícula ou inscrição.

2.1.5 Lista Consolidada

A **Lista Consolidada (LC)** é uma estrutura hierárquica de classes que representam as funções, subfunções e processos de negócio executados pela **Administração Pública (AP)**, contemplando a sua descrição e avaliação. [57]

Serve de referencial para a criação de instrumentos organizacionais ou pluriorganizacionais para a classificação e avaliação da informação pública (denominados *Tabelas de Seleção*). Além disso é incremental pelo que ao longo do tempo o número de classes irá aumentar sendo este incremento da responsabilidade da **DGLAB** que coordena e aprova a integração de novas classes.

A LC permite: [57]

- O uso de uma linguagem comum na AP;
- Determinar a entidade responsável pela conservação permanente da informação;
- Partilhar e rentabilizar a informação;
- Racionalizar e agilizar processos;
- Controlar de forma mais eficaz os diferentes ciclos de vida informacional;
- Diminuir despesas correntes.

A LC é o resultado de 3 projetos:

- Projeto MEF - Macroestrutura Funcional: deste projeto resultaram os primeiros níveis (1º e 2º) da LC (funções e subfunções de administração) permitindo uma perspetiva global e integradora do setor público;
- Projeto Harmonização de classes de 3.º nível em planos de classificação conformes à MEF: deste projeto resultou a identificação e descrição dos processos de negócio da LC (classes de nível 3);
- Projeto ASIA - Avaliação Suprainstitucional da Informação Arquivística: neste último projeto foi possível estabelecer e associar o PCA e DF a cada processo de negócio da LC.

2.1.6 Classificação

A classificação arquivística é uma operação que visa a organização e representação da informação, tendo em vista a sua contextualização e garantir a sua autenticidade e integridade. [30] Além disso é a base para a avaliação da informação, constituindo-se como condição para a eficácia e a eficiência administrativas. [30] Esta classificação é suportada por um instrumento constituído por um esquema de classes pré-definidas e por um conjunto de regras ou instruções de aplicação (plano de classificação) [30].

2.1.7 Avaliação

A avaliação arquivística é uma operação que visa a atribuição de valor à informação arquivada, para efeitos de conservação ou de eliminação, fundamentada pelo PCA e pelo DF. [30]

Tem por objetivo a implementação de boas práticas de gestão, a adequada conservação da informação que garante direitos e deveres e preserva a memória social e individual e a eliminação da informação desnecessária. [30]

A avaliação é suportada por um instrumento denominado tabela de seleção (que é construído a partir da LC).

2.1.8 Tabela de Seleção

Uma **Tabela de Seleção (TS)** é um instrumento de gestão onde se encontra: [58]

- a estrutura classificativa da informação, clarificando o seu âmbito e conteúdo (classificação)
- a definição do **Destino Final** e do **Prazo de Conservação Administrativa** e sua fundamentação (avaliação).

Uma **TS** será sempre um subconjunto de classes da **LC** para uma determinada entidade (**TS Organizacional**) ou conjunto de entidades (**TS Pluriorganizacional**) num determinado instante de tempo (a data da sua criação/aprovação). A aprovação das **TS's** é efetuada pela **DGLAB**.

Enquanto a **LC** está em constante mutação e atualização uma **TS** fica congelada no tempo. Poderão ser solicitadas alterações a uma **TS** mas carecem sempre de uma aprovação por parte da **DGLAB**.

A **CLAV** torna possível criar assistidamente as **TS's** evitando desde logo vários erros de utilizador e permitindo acelerar a criação destas.

A **TS** pode integrar [58]:

- Portaria de Gestão de Documentos, quando aplicada à documentação ativa;
- Relatório de Avaliação de Documentação Acumulada, quando aplicada às massas documentais acumuladas não contempladas em Portaria de Gestão de Documentos.

Através da aplicação de uma **TS** (classificação e avaliação) nos documentos produzidos (em suporte papel, eletrónico ou outro) de uma entidade é possível perceber que documentos devem ser conservados, e durante quanto tempo, ou eliminados. A eliminação de documentos sem necessidade de conservação permite disponibilizar meios e recursos para a conveniente gestão e conservação da documentação/informação produzida que precisa efetivamente de ser conservada de modo permanente. [6]

2.1.9 Donos e participantes de um Processo de Negócio

Os **PN's** podem ser executados por uma entidade ou várias entidades que podem ter diversos tipos de intervenção. A intervenção das entidades pode ser de dono (a entidade é responsável pela condução do **PN**, pelo produto final e por garantir a conservação da informação) ou de participante (a entidade contribui para o desenvolvimento do **PN** e do produto final).

2.1.10 Autos de Eliminação

Após serem selecionados os documentos a eliminar, aqueles cujo **PCA** já terminou e o **DF** é de eliminação, deve ser preenchido um **Auto de Eliminação (AE)**. O **AE** serve para controlar a eliminação dos documentos

das entidades e serve de prova de abate patrimonial. Além disso, deve ser transmitido à [DGLAB](#) e constitui uma garantia de transparência da ação administrativa, bem como da capacidade do Estado no cumprimento da sua missão [16].

Só após a [DGLAB](#) confirmar que não existem inconformidades no [AE](#) é que a entidade pode eliminar os documentos digitais e/ou físicos.

2.2 O PAPEL DA CLAV

A [CLAV](#) tem como objetivo servir de suporte para a classificação e a avaliação da informação pública ao facilitar a elaboração dos planos de classificação e tabelas de seleção bem como a consulta da [LC](#), das entidades, das tipologias e da legislação. Além disso, permite a edição da [LC](#) e desmaterializar os procedimentos de controlo de eliminação de informação através da recolha e análise de autos de eliminação.

2.3 INICIATIVAS SEMELHANTES À CLAV

Existem várias iniciativas semelhantes à [CLAV](#) que se descrevem resumidamente nesta secção.

2.3.1 *Iniciativa Finlandesa*

Na Finlândia usam uma ferramenta de gestão de registos chamada [AMS](#)¹ que adiciona metadados aos registos com pouca intervenção humana. Estes metadados indicam o controlo de acesso e o tempo de retenção de cada registo. Esta ferramenta identifica os registos criados e recebidos pela organização e indica como devem ser manuseados.

O *core* desta ferramenta é um esquema de classificação funcional, o que possui algumas semelhanças com a [LC](#). Assim, quando um registo é adicionado a um [Electronic Records Management System \(ERMS\)](#) os valores *default* dos metadados vêm do [AMS](#). Em alguns casos é necessário alterar o valor *default* manualmente visto que o sistema nem sempre consegue determinar as restrições de acesso de um registo.

Portanto um utilizador tem de operar tanto no esquema de classificação funcional como no [ERMS](#). Os utilizadores podem achar o esquema de classificação e o [ERMS](#) difícil de entender e de usar [19] e como tal, há a necessidade por parte da iniciativa de tornar o processo de seleção mais fácil ou automatizá-lo.

No artigo [19] é desenvolvida mais a fundo esta iniciativa dando possíveis soluções para o problema anteriormente descrito.

¹abreviação da palavra finlandesa “arkistonmuodostussuunnitelma”

2.3.2 *Iniciativa da Universidade Victoria de Toronto, Canadá*

A Universidade *Victoria* criou um esquema de classificação com o intuito de classificar os registos da sua universidade. Para tal, usa dois princípios, a classificação funcional e a classificação multi nível. O primeiro organiza os registos pelo tipo de funções (finanças, administração, recursos humanos, etc) com os quais está relacionado. Quanto à classificação multi nível, os registos são organizados por 3 níveis de classificação. No primeiro nível é dividido pelas principais funções da universidade (classificação funcional). No segundo nível subdivide-se a função nas principais atividades (por exemplo a função de finanças inclui: auditoria, taxas, contabilidade, procuração, etc). Por fim, o terceiro nível divide cada atividade num tipo de registo com as suas diretrizes específicas de retenção e de disposição (destruição ou conservação). Por exemplo, a contabilidade das finanças inclui gestão de contas, contas a pagar, etc. O terceiro nível pode possuir mais divisões dependendo do número de registos. [53]

Este esquema encontra-se ainda em desenvolvimento e apresenta-se num estado inicial e apenas aplicável a alguns casos específicos.

Se compararmos com a LC percebe-se que as duas são muito semelhantes. Mas este esquema, que se aplica a uma entidade (a Universidade *Victoria*), é ainda mais semelhante a uma TS visto que as TS's aplicam-se a uma entidade ou conjunto de entidades.

2.3.3 *Comparação com a CLAV*

As várias iniciativas aqui descritas encontram-se todas numa fase embrionária ou sem uma aplicação real sobre os documentos/registos. As várias razões para tal são o facto de possuírem demasiados processos de negócio/tipos de registo (o que torna o processo de classificação e de avaliação muito difícil ou impossível em termos práticos) ou porque ainda não chegaram a uma fase final de aplicação real.

Quanto à CLAV, apesar de a LC não se encontrar num estado final, encontra-se perto, e num estado passível de ser aplicado à vida real para várias entidades. Além disso, a CLAV possui neste momento cerca de 1000 processos de negócio e com o tempo deve chegar no máximo a cerca de 2000 processos de negócio, um número gerível de processos de negócio.

Em Portugal, a LC suportada pela CLAV será em termos legislativos o instrumento oficial para a gestão da informação das entidades públicas. Haverá um período de transição que será também auxiliado pela plataforma CLAV através da disponibilização dos RADA [7].

2.4 ESTRUTURA DA PLATAFORMA CLAV

A CLAV está dividida em duas partes:

- interface (*front-end*) acessível em <https://clav.dglab.gov.pt>;
- API de dados (*back-end* que inclui também duas bases de dados, *GraphDB* e *MongoDB*) acessível em <https://clav-api.dglab.gov.pt>.

Através da figura 1 é possível ver o possível fluxo tanto de um utilizador a aceder à interface como a de um utilizador a aceder diretamente à API de dados.

No primeiro caso, quando um utilizador acede o servidor da interface da CLAV é descarregado para o lado do utilizador o ficheiro HTML (*index*) e os vários ficheiros *JavaScript*, *CSS* e *assets* (como imagens, *PDFs*, etc) quando necessários. O servidor da interface é nada mais que um servidor *web* com recurso ao *Nginx* que hospeda estes ficheiros, os quais representam a interface construída com o *Vue* e o *Vuetify*. Como tal o código apresenta-se todo do lado do utilizador sendo os pedidos à API realizados do computador do utilizador para o servidor da API de dados sem a intervenção do servidor da interface. Ou seja, o fluxo de cada um desses pedidos será igual ao fluxo no caso em que se acede diretamente à API sem uso de qualquer interface como se pode observar na figura 1.

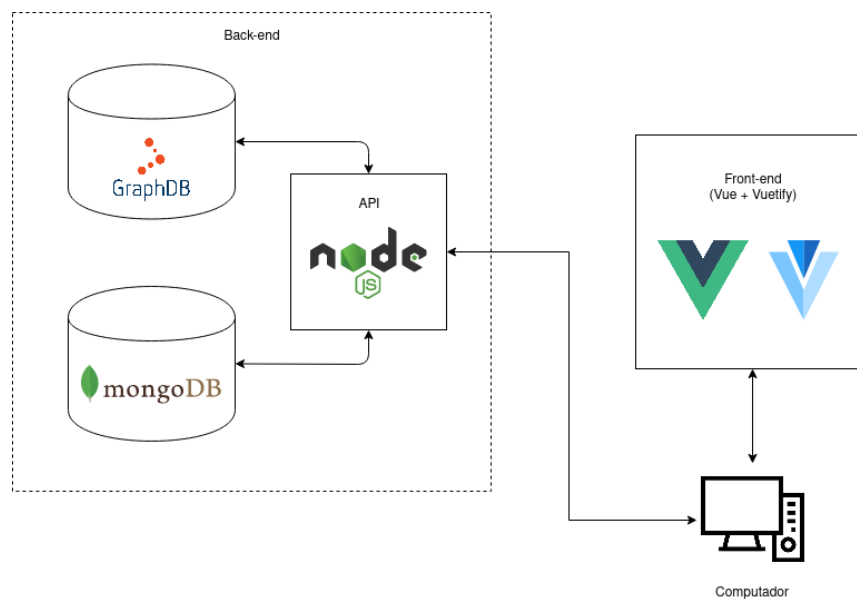


Figura 1: Estrutura da CLAV incluindo a interação de um utilizador com a mesma

Esta estrutura evoluiu depois para a estrutura presente na figura 2 em que continua a ser possível efetuar pedidos diretamente à API de dados bem como obter a interface a partir do servidor da interface. Contudo passa a ser possível fazer os pedidos à API de dados a partir do servidor da interface impedindo problemas de CORS ao efetuar pedidos a partir da interface localmente armazenada no cliente. Assim, o *Nginx* reencaminha os pedidos que são para a API de dados.

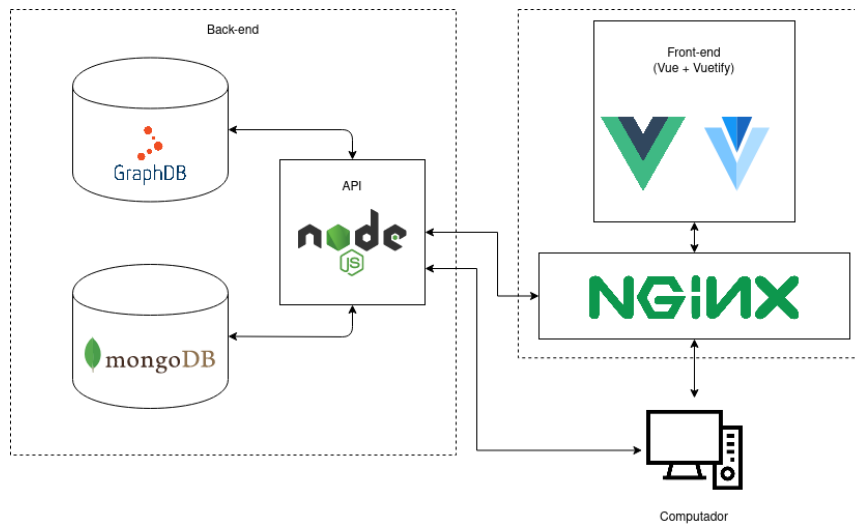


Figura 2: Estrutura evoluída da CLAV

Quanto à informação armazenada, no *GraphDB* está presente a ontologia da CLAV que contém a LC, as entidades, as tipologias, a legislação entre outros dados.

Por outro lado, no *MongoDB* são guardados os utilizadores bem como outras informações importantes para a execução da API de dados da CLAV.

2.5 FORMAS DE AUTENTICAÇÃO

A API de dados e a interface estavam inicialmente juntas (aplicação monolítica) onde as rotas eram protegidas contudo, com a separação da aplicação em duas partes, ambas partes deixaram de estar protegidas. Devido à plataforma já ter estado protegida esta já possui duas formas de autenticação, através de chaves API e através de utilizadores registados. Ou seja, tanto o registo de utilizadores e de chaves API já se encontra implementado bem como o *login* de utilizadores.

As chaves API existem por forma a dar acesso a certas rotas da API a aplicações que interajam com a mesma (por exemplo sistemas de informação) sem a necessidade de interação humana.

Já os utilizadores possuem múltiplos níveis de acesso sendo que consoante o seu nível podem ou não aceder a uma rota da interface ou da API. Os utilizadores podem autenticar-se através de *email* e *password* ou com recurso ao *Cartão de Cidadão (CC)* através do Autenticação.gov, este último apenas disponível através da interface da CLAV.

A hierarquia dos níveis de acesso na API de dados, do nível que permite menor para o maior acesso, é a seguinte:

- Nível -1: Qualquer pessoa (sem proteção),
- Nível 0: Chaves API,

- Nível 1: Representante Entidade,
- Nível 2: Utilizador Simples,
- Nível 3: Utilizador Arquivo Distrital,
- Nível 3.5: Utilizador Avançado,
- Nível 4: Utilizador Validador,
- Nível 5: Utilizador Decisor,
- Nível 6: Administrador de Perfil Funcional,
- Nível 7: Administrador de Perfil Tecnológico.

Na interface da [CLAV](#) esta hierarquia é semelhante sendo que a única diferença são os dois primeiros níveis (-1 e 0) serem substituídos por um único nível 0 que dá acesso a Qualquer pessoa.

As chaves [API](#) poderão aceder a algumas rotas com método GET. Já os utilizadores poderão realizar todos os pedidos que as chaves [API](#) podem realizar mas quanto maior o seu nível de acesso mais rotas poderão aceder.

A proteção da [API](#) de dados terá de ter esta hierarquia em conta.

2.5.1 *Registo*

Como já referido, tanto o registo de chaves [API](#) como de utilizadores já se encontra implementado.

Para o registo de uma chave [API](#) é necessário providenciar um nome, um email e a entidade a que pertence. Após o registo da chave a informação desta chave [API](#) é mantida numa base de dados *MongoDB*.

Um utilizador pode se registar através de `email + password` ou através do [Autenticação.gov](#). No primeiro caso, ao se registar necessita obviamente de indicar o seu email, a *password*, o seu nome, a entidade a que pertence e o nível de acesso que pretende. Já no caso do [Autenticação.gov](#) para o registo do utilizador é necessário todos os campos anteriores exceto a *password* (pode ser depois definida), sendo também necessário o campo [Número de Identificação Civil \(NIC\)](#) do utilizador. Caso o registo seja efetuado com recurso à interface do [Autenticação.gov](#) apenas será necessário indicar o email, a entidade a que pertence e o nível de acesso que pretende visto que os restantes campos são fornecidos pela [Autenticação.gov](#) quando o utilizador se autentica e autoriza a partilha dessa informação com a plataforma [CLAV](#). A *password* é armazenada não na sua forma literal mas sim codificada numa *hash* com a função criptográfica `bcrypt`. A utilização de funções de *hash* criptográficas ao armazenar *passwords* impede que as *passwords* originais se saibam caso a base de dados seja comprometida. Para além disso, como o

bcrypt combina um valor aleatório (*salt*) com a *password* do utilizador, é impossível pré-computar a *password* que deu origem ao *hash* sem saber o *salt*².

Durante esta tese com a proteção da API de dados ficará apenas possível o registo de utilizadores através de utilizadores que já estejam registados e possuam um nível de acesso suficiente para registar utilizadores. Estes utilizadores registados e autorizados pertencem à entidade DGLAB. Portanto, utilizadores representantes de outras entidades que queiram registar-se na plataforma terão de: [9]

- Preencher o formulário disponibilizado para o efeito, para cada representante designado pela entidade;
- O formulário deverá ser assinado por um dirigente superior da Entidade e autenticado com assinatura digital, se o envio for feito por via eletrónica (NB: não serão aceites assinaturas do formulário por dirigentes intermédios). Esta autorização autenticada pelo dirigente superior é o equivalente a uma delegação de competências, uma vez que o representante da entidade passa a ter capacidade para, em nome da entidade, submeter autos de eliminação, propostas de tabelas de seleção e novas classes para a Lista Consolidada;
- O formulário deverá ser remetido à DGLAB por via postal ou eletrónica, respetivamente, para:
 - DGLAB, Edifício da Torre do Tombo, Alameda da Universidade, 1649-010 Lisboa (formulário assinado manualmente) ou
 - clav@dglab.gov.pt (formulário com assinatura digital).
- Após receção do formulário, a DGLAB efetuará o(s) respetivo(s) registo(s) até 48 horas úteis;
- Findo esse prazo, o utilizador poderá aceder à plataforma, selecionando a opção “Autenticação”;
- A autenticação, no primeiro acesso, deve ser efetuada com o Cartão de Cidadão.

2.5.2 Login

O *login* apenas está presente para o caso dos utilizadores visto que, assim que uma chave API é registada, é enviado por email um JWT com a duração de 30 dias a ser usado nos pedidos a realizar à API de dados. O utilizador poderá ao fim dos 30 dias renovar a sua chave API, onde é gerado um novo JWT.

Portanto do lado dos utilizadores é possível como já referido realizar o *login* de duas formas através de uma estratégia local ou através do Autenticação.gov.

A estratégia local (*email* + *password*) é conseguida através do uso do *middleware Passport*. O *Passport* é um *middleware* de autenticação para *Node.js* que tem como objetivo autenticar pedidos. [43] Tem como única preocupação a autenticação delegando qualquer outra funcionalidade para a aplicação que

²Para mais informação veja *rainbow table attack*

a usa. Este *middleware* possui muitas estratégias de autenticação entre as quais a local (email/username + password), *JWT*, *OAuth*³, *Facebook* ou *Twitter*. Cada estratégia está num módulo independente. Assim as aplicações que usam o *Passport* não terão um peso adicional devido a estratégias que nem sequer usam.

No caso do *login* através do Autenticação.gov, o utilizador tem de se autenticar na interface do Autenticação.gov (a partir do botão disponível na área de autenticação da interface da *CLAV*). O fluxo do *login* antes da limitação do registo de utilizadores apenas por pessoal autorizado pode ser observado na figura 3:

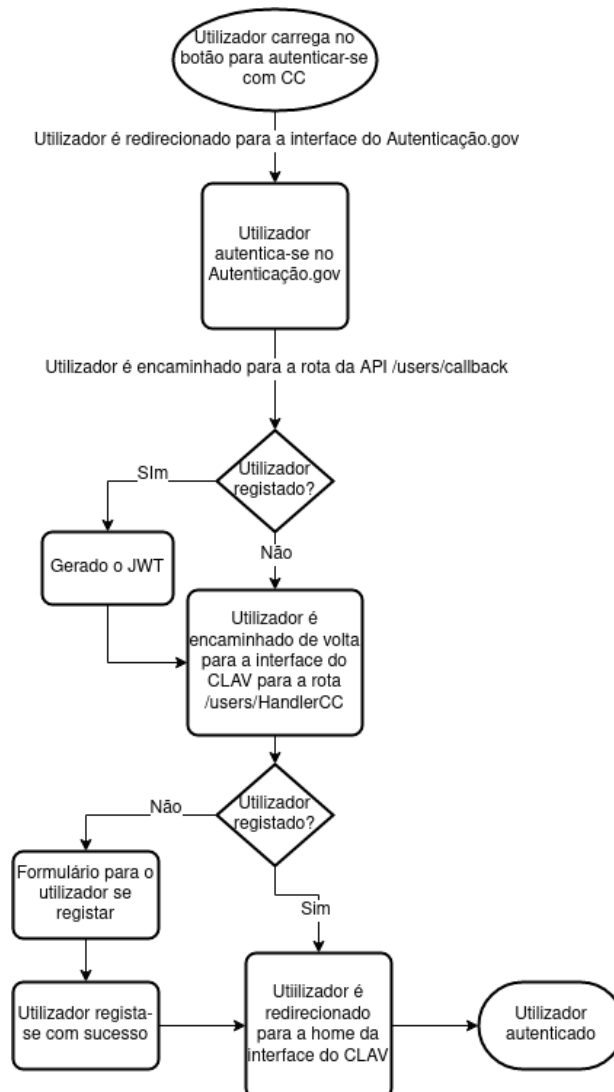


Figura 3: Fluxo do *login* de um utilizador através do Autenticação.gov antes da limitação do registo de utilizadores apenas por pessoal autorizado

Após a limitação do registo de utilizadores apenas por pessoal autorizado continuará a ser possível o utilizador autenticar-se pelo Autenticação.gov mas deixará de ser possível o utilizador registar-se aparecendo, em vez de um formulário de registo, um aviso de que o utilizador não se encontra registado.

³Protocolo *open-source* com o objetivo de permitir a autenticação simples, segura e padrão entre aplicações móveis, *web* e *desktop*

No *login* do utilizador é gerado um **JWT** com a duração de 8 horas que deve ser usado nos pedidos a realizar à **API** de dados. No fim das 8 horas o utilizador necessita de se autenticar de novo.

2.6 RESUMO

Neste capítulo foi brevemente abordada a **CLAV**, explicando os principais conceitos desta que irão ajudar a compreender algumas das secções seguintes desta dissertação.

Além disso, foi efetuada uma breve descrição de iniciativas semelhantes com a **CLAV** comparando-as a esta.

Por fim, foi descrito a estrutura e as formas de autenticação (*registo* e *login*) da **CLAV** no momento que esta dissertação iniciou com o objetivo de o leitor perceber que trabalho tem de ser efetuado e a razão de algumas decisões e implementações efetuadas nesta dissertação.

PROTEÇÃO DA API DE DADOS

A proteção da [API](#) de dados é bastante importante já que impede que utilizadores não autorizados por esta efetuem alterações nos dados fornecidos por esta [API](#) mantendo esta informação correta e consistente. Assim apenas utilizadores autorizados poderão ler, adicionar, editar ou eliminar dados da [API](#) de dados entre os quais dados de utilizadores, de entidades, de tipologias, etc.

Além disso a necessidade de múltiplos níveis de acesso deve-se ao facto de que os utilizadores da [API](#) não devem poder efetuar as mesmas operações na [API](#) de dados. Ou seja, os utilizadores devem ter diferentes permissões onde alguns só poderão ler, outros ler e adicionar, outros ler e adicionar alguns dados, etc.

Na secção seguinte é aprofundado o [JWT](#), uma tecnologia que permite auxiliar no processo de proteção da [API](#) de dados ao tornar possível que os pedidos autorizados (com um [JWT](#) válido) pela [API](#) de dados não sejam bloqueados e que aqueles sem autorização nem permissão dada pela [API](#) de dados sejam bloqueados.

3.1 ESTADO DA ARTE

3.1.1 *JSON Web Token (JWT)*

O [JWT](#) é um *open standard*¹ que define uma forma compacta e independente de transmitir com segurança informação entre partes com um objeto [JSON](#). [5]

O [JWT](#) pode ser assinado digitalmente ([JWS](#)), encriptado ([JWE](#)), assinado e depois encriptado ([JWS](#) encriptado, ou seja, um [JWE](#), ordem recomendada²) ou encriptado e depois assinado ([JWE](#) assinado, ou seja, um [JWS](#)).

Caso seja assinado digitalmente é possível verificar a integridade da informação mas não é garantida a sua privacidade contudo podemos confiar na informação do [JWT](#). A assinatura pode ser efetuada através de um segredo usando por exemplo o algoritmo [HMAC](#) ou através de pares de chaves pública/privada usando

¹Mais informação em <https://tools.ietf.org/html/rfc7519>

²Mais informação em <https://tools.ietf.org/html/rfc7519#section-11.2>

por exemplo o algoritmo **RSA**. No caso de se usar pares de chaves pública/privada a assinatura também garante que a parte envolvida que tem a chave privada é aquela que assinou o **JWT**.

Por outro lado, os **JWTs** podem ser encriptados garantindo a privacidade destes, escondendo a informação das partes não envolvidas. Nesta secção apenas se falará sobre **JWTs** e **JWSs** (**JWT** assinado). Se pretender saber mais sobre **JWEs** pode ler o capítulo 5 do livro *The JWT Handbook* por *Sebastián E. Peyrott*.

Sendo assim, em que casos é útil o uso de **JWTs**?

Dois dos casos são os seguintes:

- **Autorização:** Este será o caso para o qual o **JWT** será usado na **CLAV**. Quando o utilizador realiza o *login* gera-se um **JWT** por forma a que os restantes pedidos desse utilizador sejam realizados com esse **JWT** (**Single Sign On**). O uso de **JWTs** para estes casos permitem um *overhead* pequeno e a flexibilidade de serem usados em diferentes domínios;
- **Troca de informação:** No caso de troca de informação entre duas partes os **JWTs** assinados são de bastante utilidade visto que permitem verificar se o conteúdo não foi violado e, no caso de se usar pares de chaves pública/privada para assinar, permitem ter a certeza que o remetente é quem diz ser.

Estrutura do JWT

Os **JWTs** são construídos a partir de três elementos, o *header* (objeto **JSON** também conhecido por **JOSE header**), o *payload* (objeto **JSON**) e os dados de assinatura/encriptação (depende do algoritmo usado). Estes elementos são depois codificados em representações compactas (**Base64 URL-safe**³). As codificações **Base64 URL-safe** de cada elemento são depois concatenadas numa *string* onde é usado o carácter '.' para separar as partes, dando origem a uma representação final compacta do **JWT** (**JWS/JWE Compact Serialization**).

Na secção 3.1.1 estão presentes dois diagramas referentes à construção de dois **JWTs** sendo um deles assinado.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ1eW11IjoiaSm9zw6kgTWYydGlucyIsIm51bSI6ImE3ODgyMSJ9.
tRPSYVsFI-nziRPuAjdGZLN2tUez5MtLML_aAnPplgM

Figura 4: Exemplo de representação compacta de **JWT** (quebra de linhas por forma a melhorar leitura)

De seguida vamos aprofundar cada elemento referido:

Header: O cabeçalho (a vermelho na figura 4) é constituído pelos seguintes atributos:

³Variante da codificação **Base64** onde a codificação gerada é segura para ser usada em *URLs*. Basicamente para a codificação **Base64** gerada substitui os caracteres '+' e '/' pelos caracteres '.' e '_' respetivamente. Além disso, remove o carácter de *padding* e proíbe separadores de linha

- O atributo obrigatório `alg` (algoritmo), único campo obrigatório para o caso de um `JWT` não encriptado, onde é indicado que algoritmo é usado para assinar e/ou descriptar. O seu valor pode ser, por exemplo, `HS256` (`HMAC` com o auxílio do `SHA-256`⁴) ou `RSA`;
- O atributo opcional `typ` (tipo do `token`) em que o seu valor é `"JWT"`. Serve apenas para distinguir os `JWTs` de outros objetos que têm um `JOSE header`;
- O atributo opcional `cty` (tipo do conteúdo (`payload`)). Se o `payload` contiver atributos arbitrários este atributo não deve ser colocado. Caso o `payload` seja um `JWT`⁵ então este atributo deve ter o valor de `"JWT"`.

O cabeçalho é de grande importância visto que permite saber se o `JWT` é assinado ou encriptado e de que forma o resto do `JWT` deve ser interpretado.

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

Exemplo 3.1: `Header` usado para construir o `JWT` da figura 4

Payload: O `payload` (a roxo na figura 4) contém a informação/dados que pretendemos transmitir com o `JWT`. Não há atributos obrigatórios contudo existem certos atributos que têm um significado definido (atributos registados).

Existem 7 atributos registados (*registered claims*): [45]

- `iss` (*issuer*): Identificador único (*case-sensitive string*) que identifica unicamente quem emitiu o `JWT`. A sua interpretação é específica de cada aplicação visto que não há uma autoridade central para gerir os emissores;
- `sub` (*subject*): Identificador único (*case-sensitive string*) que identifica unicamente de quem é a informação que o `JWT` transporta. Este atributo deve ser único no contexto do emissor, ou se tal não for possível, globalmente único. O tratamento do atributo é específico a cada aplicação;
- `aud` (*audience*): Identificador único (*case-sensitive string*) ou *array* destes identificadores únicos que identificam unicamente os destinatários pretendidos do `JWT`. Ou seja, quem lê o `JWT` se não estiver no atributo `aud` não deve considerar os dados contidos no `JWT`. O tratamento deste atributo também é específico a cada aplicação;

⁴Função pertencente ao conjunto de funções *hash* criptográficas `Secure Hash Algorithm 2 (SHA-2)` desenhadas pela NSA

⁵`JWT` aninhado (*nested JWT*)

- `exp` (*expiration (time)*): Um número inteiro que representa uma data e hora específicos no formato *seconds since epoch* definido pela [POSIX](#)⁶, a partir da qual o `JWT` é considerado inválido (expira);
- `nbf` (*not before (time)*): Representa o inverso do atributo `exp` visto que é um número inteiro que representa uma data e hora específicos no mesmo formato do atributo `exp`, mas a partir da qual o `JWT` é considerado válido;
- `iat` (*issued at (time)*): Um número inteiro que representa uma data e hora específicos, no mesmo formato dos atributos `exp` e `nbf`, que representa o instante no qual o `JWT` foi emitido;
- `jti` (*JWT ID*): Identificador único (*string*) do `JWT` que permite distinguir `JWTs` com conteúdo semelhante. A implementação tem de garantir a unicidade deste identificador.

Estes atributos registados têm todos 3 caracteres visto que um dos requisitos do `JWT` é ser o mais pequeno/compacto possível.

Existem depois mais dois tipos de atributos, públicos e privados. Os atributos públicos podem ser definidos à vontade pelos utilizadores de `JWTs` mas têm de ser registados em *IANA JSON Web Token Claims registry* ou definidos por um espaço de nomes resistente a colisões de forma a evitar a colisão de atributos. Já os atributos privados são aqueles que não são nem registados nem públicos e podem ser definidos à vontade pelos utilizadores de `JWTs`. Os dois atributos usados no exemplo 3.2 (`name` e `num`) são atributos privados.

```
{
  "name": "José Martins",
  "num": "a78821"
}
```

Exemplo 3.2: *Payload* usado para construir o `JWT` da figura 4

Signature: A assinatura (a azul na figura 4) é criada ao usar o algoritmo indicado na *header* no atributo `alg` tendo como um dos argumentos os elementos codificados da *header* e do *payload* juntos por um ponto e como outro argumento um segredo. O resultado do algoritmo é depois codificado em Base64 URL - safe. Esta assinatura no caso dos `JWSs` é usada para verificar a integridade do `JWT` e caso seja assinado com uma chave privada permite também verificar se o remetente é quem diz ser. No caso de o atributo `alg` for `none` a assinatura é uma `string` vazia.

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  segredo1.-uminho!clav
)
```

⁶Mais informação em https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16

Exemplo 3.3: *Signature* usado para construir o *JWT* da figura 4

Criação de *JWT/JWS*

Na figura 5 é apresentada a construção de um *JWT* em que o atributo *alg* (algoritmo) tem o seu valor igual a *none*, ou seja, o *JWT* não é assinado nem encriptado.

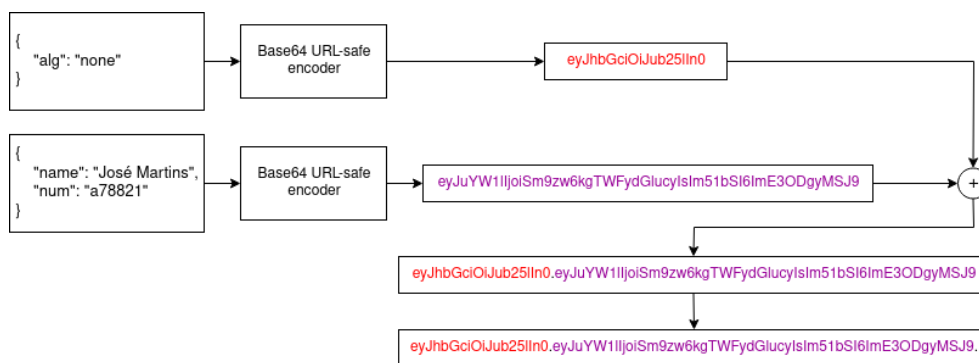


Figura 5: Criação de um *JWT*

Já na figura 6 é demonstrada a construção de um *JWT* assinado, ou seja, um *JWS*.

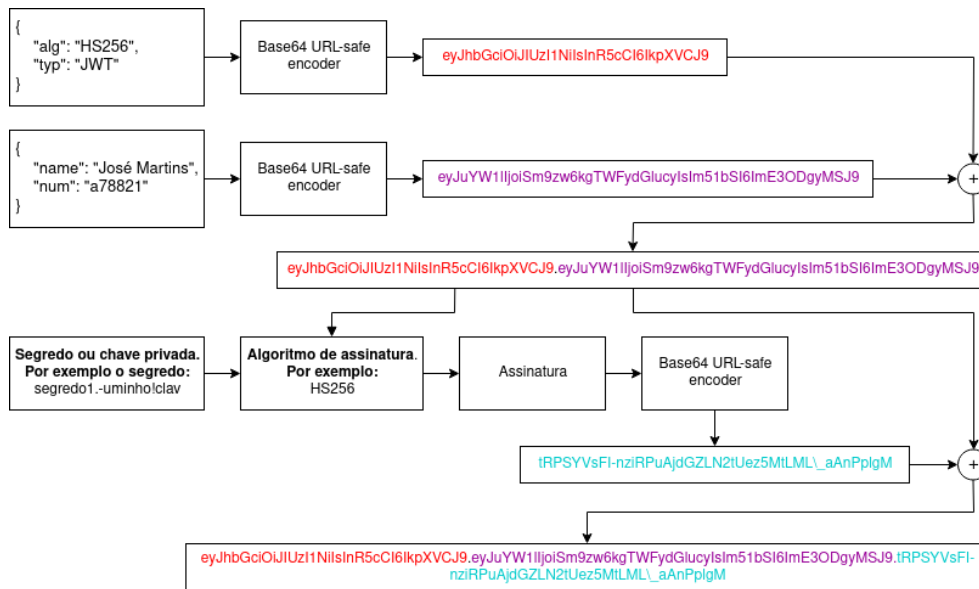


Figura 6: Criação de um *JWS*

Alternativas ao JWT

Algumas alternativas ao JWT passam pelo uso de Simple Web Token (SWT) ou Security Assertion Markup Language (SAML). Se compararmos o JWT ao SAML, o JSON é menos verboso que o XML e mesmo quando codificado o seu tamanho é menor.

De um ponto de vista de segurança o SWT apenas pode ser assinado simetricamente por um segredo partilhado usando o algoritmo HMAC. Já o JWT e o SAML podem usar pares de chaves pública/privada para assinar. Contudo assinar XML com XML Digital Signature sem introduzir buracos de segurança é mais difícil quando comparado com a simplicidade de assinar JSON. [5]

Houve contudo algumas bibliotecas de JWT com vulnerabilidades devido ao atributo `alg` da `header` do JWT. Havia duas situações de vulnerabilidade:

- As bibliotecas ao fazer a verificação (recebe um JWT e um segredo/chave pública como argumentos) de um JWT com `alg` igual a `none` assumiam logo que o JWT era válido mesmo que o segredo/chave pública fosse diferente de vazio. Ou seja, com a simples alteração do atributo `alg` e com a remoção da `signature` podia-se alterar o `payload` do JWT que o servidor iria continuar a considerar que a integridade do JWT não foi colocada em causa mesmo que os JWTs gerados pelo servidor tivessem sido assinados com um algoritmo e com recurso a um segredo/chave privada;
- As bibliotecas ao fazer a verificação seja de um algoritmo simétrico ou assimétrico apenas tinham como parâmetros o JWT e o segredo/chave pública. Isto gera uma segunda vulnerabilidade. Se o servidor estiver à espera de um JWT assinado com pares de chaves pública/privada mas receber um JWT assinado com HMAC vai assumir que a chave pública é o segredo a usar no algoritmo HMAC. Ou seja, se se criar um JWT com o atributo `alg` igual a `HMAC` e a assinatura for gerada usando o algoritmo HMAC com o segredo a ser a chave pública, podemos alterar o `payload` (antes de assinar) que o servidor vai considerar que o JWT não foi maliciosamente alterado.

Portanto a flexibilidade de algoritmos dada pelo JWT coloca em causa a segurança pelo que da parte das bibliotecas o atributo `alg` não deve ser considerado [37] bem como deve ser *deprecated* e deixar de ser incluído nos JWTs⁷.

A biblioteca que será usada na CLAV, `jsonwebtoken`⁸, já endereçou estes problemas⁹ pelo que estas vulnerabilidades não estarão presentes na CLAV.

Ainda comparando as diferentes alternativas, os `parsers` de JSON são mais comuns em grande parte das linguagens de programação visto que os JSONs mapeiam diretamente para objetos ao contrário do XML que não tem um mapeamento natural de documento para objeto. [5] Isto torna mais fácil trabalhar com JWT do que com SAML.

⁷Ver <https://gist.github.com/paragonie-scott/c88290347c2589b0cd38d8bb6ac27c03>

⁸Ver <https://www.npmjs.com/package/jsonwebtoken>

⁹Ver <https://github.com/auth0/node-jwt-token/commit/1bb584bc382295eeb7ee8c4452a673a77a68b687>

Já quando comparamos os [JWTs](#) a *cookie sessions*, o [JWT](#) tem a vantagem de as sessões puderem ser *stateless* enquanto que as *cookies* são *statefull*. Contudo, ser *stateless* não permite por exemplo que a qualquer altura se possa revogar um [JWT](#). Para endereçar esse problema é necessário, por exemplo, guardar (*statefull*) os [JWTs](#) numa base de dados associando cada [JWT](#) ao identificador único de quem é a informação contida no [JWT](#) (o uso de uma *whitelist*). Assim para revogar um [JWT](#) bastaria removê-lo da base de dados.

Outra alternativa ao [JWT](#) seria *sessionIDs*. As *sessionIDs* são *strings* longas, únicas e aleatórias. É possível revogar um *sessionID*, ao contrário do [JWT](#), bastando para isso remover o *sessionID* da base de dados.

Por fim, uma outra alternativa bastante semelhante ao [JWT](#) é *Branca*. *Branca* usa o algoritmo simétrico [IETF XChaCha20-Poly1305 AEAD](#) que permite criar *tokens* encriptados com a garantia de integridade. Tem também uma região de *payload* como o [JWT](#) com a única diferença é que este *payload* não tem uma estrutura definida. Não necessita da *header* visto que o algoritmo usado não varia. Em vez de usar codificação em Base64 URL - safe usa Base62 que também é *URL-safe*. Para além disso o *token* gerado é geralmente de menor dimensão do que o gerado pelo [JWT](#) sendo como tal mais compacto que o [JWT](#). [52] Visto que o *Branca* encripta e garante integridade de uma forma mais simples que o [JWT](#) permite (para isso era necessário recorrer a um [JWE](#) que tem no seu *payload* um [JWS](#)), sendo como tal propenso a menos erros de programação. Contudo, o *Branca* ainda não é muito conhecido nem um *standard* da indústria, ao contrário do [JWT](#), mas não deixa de ser algo a ter em conta para o futuro.

3.2 SOLUÇÃO

A proteção da [API](#) de dados é bastante importante visto que impede o uso indevido de pessoal não autorizado, isto é, não registado. Para que um utilizador possa aceder à [API](#) de dados necessita de criar uma Chave [API](#) ou de pedir o registo de uma conta.

A proteção da [API](#) de dados da [CLAV](#) possui os seguintes requisitos:

- Todos os utilizadores devem conseguir aceder às rotas a que as Chaves [API](#) dão acesso;
- Deve ser possível definir para cada rota quem pode aceder, Chaves [API](#) e/ou utilizadores;
- Se numa rota apenas podem aceder utilizadores deve ser possível definir que níveis de utilizadores podem aceder a essa rota;
- A verificação da autorização e autenticação de um pedido a uma rota é realizada a partir de um *token* ([JWT](#));
- Tanto uma Chave [API](#) como um *token* de utilizador são um [JWT](#);

- Uma Chave **API** para além de ainda ser válida (não ter expirado) deve estar ainda ativa;
- Um utilizador desativado não pode gerar um *token* de utilizador (não pode realizar *login*);
- Uma Chave **API** deve ter a validade de 30 dias;
- Um *token* de um utilizador deve ter a validade de 8 horas;
- A **API** deve conseguir distinguir uma Chave **API** de um *token* de utilizador;
- Não deve ser possível um utilizador fazer-se passar por uma Chave **API** e vice-versa.

Os pedidos a efetuar à **API** devem possuir o **JWT** na *header HTTP Authorization* ou na *query string* do pedido num dos seguintes campos:

token caso seja o token de um utilizador:

`http://example.com/path/page?token=<token>`

apikey caso seja uma Chave **API**:

`http://example.com/path/page?apikey=<Chave API>`

Na *header Authorization* é usado o esquema de autenticação *Bearer*¹⁰ com umas pequenas alterações. Portanto o conteúdo da *header Authorization*:

- Caso seja o token de um utilizador é:
`token <token>`
- Caso seja uma Chave **API** é:
`apikey <Chave API>`

ao invés do esquema de autenticação predefinido do *Bearer*: `Bearer <token/Chave API>`.

Caso não seja respeitado nenhum destes formatos, o pedido será recusado visto que a **API** de dados não consegue verificar o *token*.

Com estas duas formas de enviar o **JWT** é possível distinguir os utilizadores das Chaves **API** e cumprir assim um dos requisitos. Além disso, esta divisão entre utilizadores e chaves **API** permite uma mais fácil gestão dos *tokens* recebidos pela **API** bem como permite usar duas formas diferentes de os gerar/verificar com o possível benefício de melhorar a segurança da **API** já que também permite usar dois pares de chaves pública/privada, uma para as Chaves **API**'s e outra para os *token*'s de utilizadores impedindo assim que um utilizador se faça passar por uma chave **API** e vice-versa. Para cada par de chaves geram-se os *tokens* usando a chave privada e validam-se usando a chave pública.

¹⁰Mais informação em <https://tools.ietf.org/html/rfc6750>

Um dos requisitos é que os *tokens* gerados pela API sejam *JWTs*. Contudo poderiam ser de outro tipo, por exemplo, uma *string* aleatória e única. Neste caso, o processo de envio dos *tokens* para a API manter-se-ia igual. Apenas seria alterada a forma de geração e verificação do *token*, ou seja, uma alteração interna invisível para o utilizador.

De seguida é apresentado um diagrama com a estratégia de proteção que será implementada para cumprir os requisitos anteriormente enunciados:

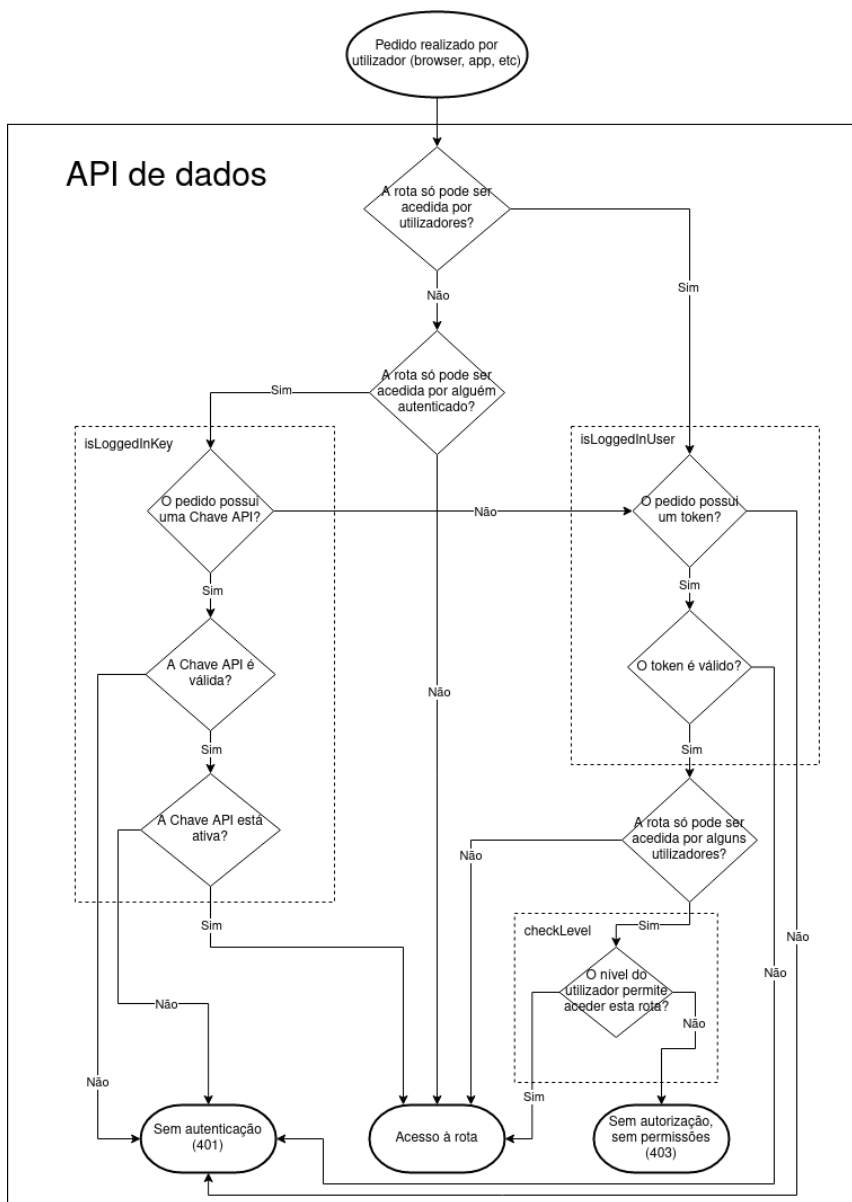


Figura 7: Estratégia de proteção da API de dados

Como se pode observar na figura 7, o `isLoggedInKey`, o `isLoggedInUser` e o `checkLevel` serão *middlewares* que caso sejam usados numa rota definem desde logo quem pode aceder. Ou seja, se for

usado o `isLoggedInUser` sabe-se desde logo que apenas os utilizadores podem aceder. Por outro lado, se for usado o `isLoggedInKey` significa que podem aceder à rota as Chaves API e os utilizadores. Para além disso, quando o `checkLevel` é usado após o `isLoggedInUser`, sabe-se que apenas parte dos utilizadores pode aceder, sendo que este *middleware* recebe como argumento os níveis de utilizadores que podem aceder à rota. A informação do nível do utilizador está presente no JWT (*token*) enviado. Se não for usado nenhum dos *middlewares* significa que qualquer pessoa pode aceder a rota.

Cumprem-se assim, quase todos os requisitos com a exceção dos da validade dos *tokens*. Estes são atingidos na geração dos JWT's no qual o *token* de um utilizador é gerado para ter apenas a validade de 8 horas e uma Chave API é gerada para ter a validade de 30 dias.

Após descrito como devem ser feitos os pedidos à API e a estratégia de proteção a desenvolver, irão ser apresentados possíveis fluxos de interação entre utilizadores (*browser, app, etc*) e o servidor da API.

O fluxo de autenticação de um utilizador na API a ser implementado pode observar-se na figura 8:

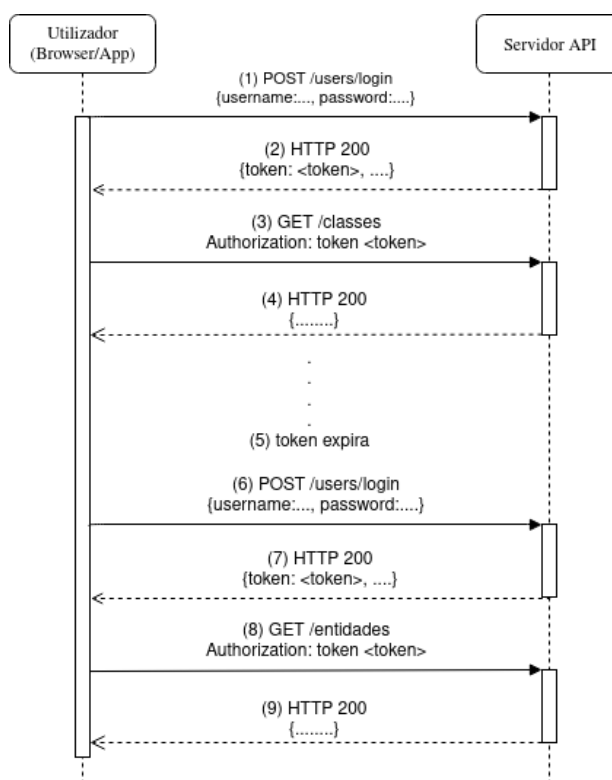


Figura 8: Fluxo de autenticação e posteriores pedidos de um utilizador

1. Utilizador autentica-se ao providenciar o seu *email* e a sua *password*;
2. Caso o utilizador se autentique com sucesso é devolvido um *token* que deve ser usado nos restantes pedidos até expirar;
3. Utilizador realiza um pedido para obter as classes, colocando o *token* na *header Authorization*;

4. Caso o *token* enviado seja válido e não tenha expirado são devolvidas as classes;
5. *Token* expirou após o tempo definido;
6. Utilizador realiza uma nova autenticação por forma a obter um novo *token*;
7. Caso o utilizador se autentique com sucesso é devolvido um *token* que deve ser usado nos restantes pedidos até expirar;
8. Utilizador realiza um pedido para obter as entidades, colocando o *token* na *header Authorization*;
9. Caso o *token* enviado seja válido e não tenha expirado são devolvidas as entidades.

O fluxo de autenticação e renovação de uma Chave API na API a ser implementado pode ser observado na figura 9:

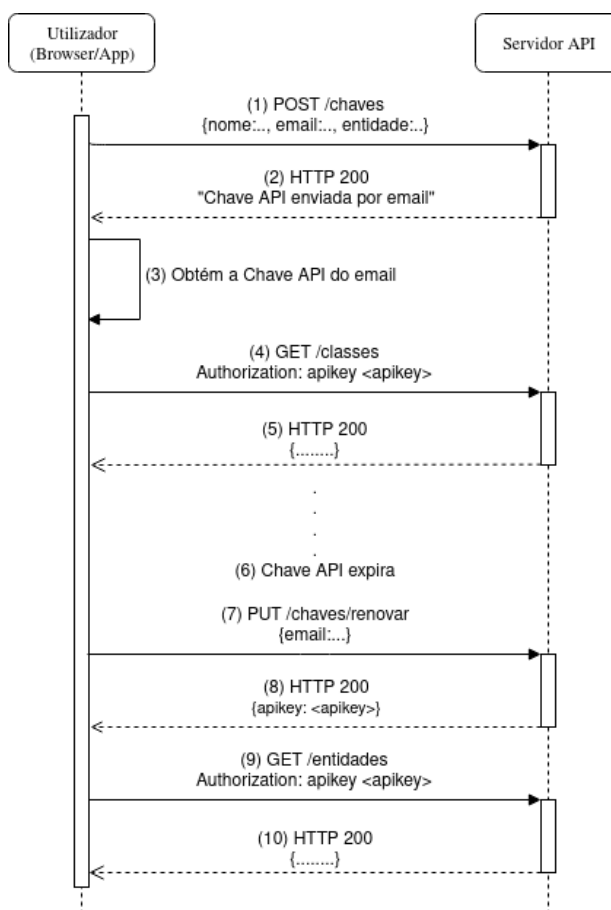


Figura 9: Fluxo de autenticação e posteriores pedidos de uma chave API

1. Utilizador cria uma chave API ao providenciar o nome, o email e a entidade a que pertence;

2. A Chave API é enviada para o email fornecido pelo utilizador com o objetivo de ser usada nos próximos pedidos;
3. O utilizador obtém a chave API no email enviado;
4. Utilizador realiza um pedido para obter as classes, colocando a chave API na *header Authorization*;
5. Caso a Chave API enviada seja válida e não tenha expirado são devolvidas as classes;
6. Chave API expirou após o tempo definido;
7. Utilizador renova a Chave API ao providenciar o email usado para criar a Chave API;
8. A nova (renovada) Chave API é devolvida para ser usada nos restantes pedidos;
9. Utilizador realiza um pedido para obter as entidades, colocando a Chave API na *header Authorization*;
10. Caso a Chave API enviada seja válida e não tenha expirado são devolvidas as entidades.

3.3 IMPLEMENTAÇÃO

Para proteger as rotas da API é necessário haver métodos de verificação dos *tokens* com o objetivo de decidir se o utilizador/Chave API pode aceder a uma determinada rota. Isto é perceptível na imagem 7 onde se destacam 3 *middlewares*, *isLoggedInKey*, *isLoggedInUser* e *checkLevel*. De seguida será apresentado o pseudo-código destes *middlewares*.

Por forma a validar se uma Chave API pode aceder a uma determinada rota é executado o *middleware isLoggedInKey*:

```
function isLoggedInKey(req, res, next)
  key = getJWTfromHeaderOrQueryString('apikey')

  if key then
    keyBD = getKeyFromMongoDB(key)
    if keyBD then
      res = jwt.verify(key, publicKeyForAPIkey, { algorithms: ["RS256"] })
      if res != expired then
        if keyBD.active == True then
          return next()
        else
          #HTTP status 403
          return "API Key disabled"
      else
        #HTTP status 401
        return "Unauthorized"
    else
      #HTTP status 401
```

```

    return "Unauthorized"
  else
    return isLoggedInUser(req, res, next)

```

Exemplo 3.4: Verificação se um pedido com uma determinada Chave [API](#) pode ser efetuado

É importante destacar a chamada da função `isLoggedInUser` que é executada no caso de não ser detetado uma Chave [API](#) no pedido (na *header Authorization* ou na *query string apikey*) e como tal, com essa chamada, tenta-se perceber se afinal foi passado um *token* de um utilizador já que todos os utilizadores conseguem aceder às rotas que as Chaves [API](#) conseguem como já referido.

No seguimento, para validar se um determinado *token* de um utilizador pode aceder a uma determinada rota é executado o *middleware* `isLoggedInUser`:

```

JWTstrategy = passport-jwt.Strategy

passport.use("jwt", new JWTstrategy(
  secretOrKey: publicKeyForUser,
  algorithms: ["RS256"],
  jwtFromRequest: getJWTfromHeaderOrQueryString('token')
, (token, done) => done(null, token)))

function isLoggedInUser(req, res, next)
  passport.authenticate("jwt", { session: false }, function (err, user, info)
    if err then
      #HTTP status 401
      return "Unauthorized"
    if !user then
      #HTTP status 401
      return "Unauthorized"
    req.logIn(user, function(err)
      if err then
        #HTTP status 401
        return "Unauthorized"

      next()
    )
  )(req, res, next)

```

Exemplo 3.5: Verificação se um pedido com um determinado *token* de um utilizador registado pode ser efetuado

Os *tokens* tanto das Chaves [API](#) como de *tokens* de utilizadores são obtidos através da utilização de extratores presentes na estratégia `passport-jwt` do `passport`. Assim para extrair o *token* da *query string* basta:

```

var ExtractJWT = require("passport-jwt").ExtractJwt
token = ExtractJWT.fromUrlQueryParameter("<nome do campo, 'token' ou 'apikey' no caso da CLAV>")

```

Exemplo 3.6: Extração do *token* da *query string*

Já para extrair o *token* da *header Authorization* basta:

```
var ExtractJWT = require("passport-jwt").ExtractJwt
token = ExtractJWT.fromAuthHeaderWithScheme("<palavra antes do token, 'Bearer' no caso dum bearer token, 'token'
ou 'apikey' no caso da CLAV>")
```

Exemplo 3.7: Extração do *token* da *header Authorization*

Para verificar se o utilizador registado tem um nível suficiente para aceder a uma rota, depois de se verificar que o utilizador está autenticado (*isLoggedInUser*), deve-se executar o *middleware checkLevel*:

```
function checkLevel(clearance)
  return function(req, res, next)
    havePermissions = False

    if clearance is Array then
      if req.user.level in clearance then
        havePermissions = True
    else
      if req.user.level >= clearance then
        havePermissions = True

    if havePermissions then
      return next()
    else
      #HTTP status 403
      return "Without enough permissions"
```

Exemplo 3.8: Verificação se um utilizador registado tem permissões suficientes para aceder a uma determinada rota

Ou seja, a variável *clearance* poderá ser uma lista de números ou apenas um número. No primeiro caso verifica-se que o nível do utilizador está presente na lista, em caso afirmativo então o utilizador tem permissões para aceder. Já no segundo caso, o utilizador só terá permissões para aceder se o seu nível foi igual ou superior ao *clearance*.

Com estes três *middlewares* é possível proceder à proteção da *API* da *CLAV* garantindo que utilizadores com diferentes níveis de acesso apenas conseguem aceder ao que lhes é permitido.

3.3.1 Interface da *CLAV*

A interface tem vários objetivos, um deles é a disponibilização de várias informações de forma pública. Com a proteção da *API* de dados, é obrigatório em quase todas as rotas o uso de uma Chave *API* ou de um *token* de utilizador o que impossibilita a disponibilização de dados de forma pública a partir da interface. Para contornar este obstáculo, criou-se uma Chave *API* específica para a interface para esta puder realizar pedidos à *API*. O único problema agora é, como a interface de cada cliente irá obter a Chave *API*?

A solução passa pela criação de uma rota específica na API de dados (GET /chaves/clavToken) que caso o Origin do pedido seja uma das interfaces da CLAV devolve a Chave API da CLAV. Esta rota internamente, cria a Chave API caso não exista, renova-a se já tiver expirado e por fim devolve-a. Assim apenas permite-se a obtenção desta Chave API pelas interfaces (através do cabeçalho Origin) e permite-se disponibilizar na interface de forma aberta várias informações obtidas a partir da API. A Chave API na interface é armazenada em *localStorage*. Na API de dados há uma variável, *interfaceHosts*, para definir os domínios válidos das interfaces.

Para além da proteção na API de dados é necessário a proteção na interface com o objetivo de impedir o acesso indevido de utilizadores a certas páginas da interface, naquelas em que não lhe são destinadas por alguma razão bem como aquelas em que um dos pedidos à API de dados irá falhar por falta de permissões. Para tal, como a interface é criada em *Vue.js* podemos associar a cada rota da interface (página) um meta valor *levels* com os níveis de quem pode aceder. Estes níveis vão de 0 a 7 e o 0 indica que qualquer pessoa pode aceder (sem proteção ou Chaves API) e de 1 a 7 são os níveis de utilizadores iguais aos presentes na proteção da API de dados. Com este meta valor, sempre que a rota da interface muda é verificado se o utilizador pode aceder à rota, ou seja, se tem autenticação (caso necessário) e/ou autorização (caso necessário). Caso não tenha autorização o utilizador é redirecionado para a página inicial da interface e é devolvida uma mensagem de falta de permissões, mantendo o utilizador autenticado. Já no caso de falta de autenticação o utilizador é redirecionado para a página de autenticação e devolve uma mensagem de falta de autenticação tendo como possíveis razões não estar autenticado ou o *token* ter expirado. Convém acrescentar que o *token*, o nome e a entidade do utilizador são armazenados em *localStorage*.

Ainda na interface foram feitas algumas melhorias de segurança e performance na realização dos pedidos à API de dados. Por forma a evitar com antecedência pedidos que já se sabe que irão falhar por falta de autenticação o que se faz é verificar o *token* antes de efetuar qualquer pedido à API de dados. Isto é possível porque para gerar os *tokens* são usados pares de chaves pública/privadas. Assim, para realizar esta verificação a interface apenas necessita de ter as chaves públicas. Em termos técnicos foi criado um *plugin Vue.js* que acrescenta o método *request* à *framework* que deve ser usado para efetuar os pedidos à API de dados, sendo que este método trata de tudo o que é necessário para realizar os pedidos, desde obter o *token* do *localStorage*, colocar o *token* de forma apropriada no pedido a efetuar, bem como trata da verificação do *token* antes de efetuar o pedido. No caso do *token* de um utilizador expirar este é redirecionado para a página de autenticação enquanto que no caso da Chave API da interface expirar pede de novo a Chave API à API pela rota anteriormente referida (GET /chaves/clavToken). Depois do pedido ser realizado, este método também faz *parse* de parte dos erros, ou seja, quando a resposta do pedido tem HTTP status 401 ou 403 este método redireciona o utilizador e devolve uma mensagem de erro de acordo com o HTTP status. Portanto quando é 401 e o utilizador estava autenticado o utilizador é redirecionado para a página de autenticação com uma mensagem de falta de autenticação. Se for uma Chave API é redirecionado para a página inicial com uma mensagem de erro para tentar de novo. Já

quando é 403, este caso apenas acontece com utilizadores autenticados, o utilizador é redirecionado para a página inicial com uma mensagem de erro de falta de permissões, mantendo o utilizador autenticado. No caso de o `HTTP status` não ser um destes o erro é devolvido à função que chamou este método. Assim quem desenvolve a interface não precisa de se preocupar com estas situações nem de acrescentar sempre manualmente o `token` ao pedido. Necessitam apenas de enviar os dados referentes ao pedido a efetuar (verbo, caminho, `body`, `query string`, etc) pelo que diminui a ocorrência de erros e facilita a manutenção do código.

Por fim, uma pequena nota, um utilizador estar ou não autenticado na interface depende apenas se o `token` do utilizador ainda se encontra ou não em `localStorage`. Em casos de `HTTP status 401` este `token` é eliminado de `localStorage` necessitando o utilizador de voltar a se autenticar para voltar a obter um `token` e colocá-lo em `localStorage`.

3.4 RESUMO

Neste capítulo é descrito como é alcançado o objetivo de proteger a `API` de dados da `CLAV` com múltiplos níveis de acesso. Para tal, descreveu-se a tecnologia usada, `JWT`, passando pela descrição da solução e sua implementação onde se descreveu a proteção da interface e como esta melhora o desempenho da `API` de dados e da própria interface.

AUTENTICAÇÃO ATRAVÉS DE CMD

Outro dos objetivos da dissertação é a autenticação com [Chave Móvel Digital \(CMD\)](#) pelo que é necessário adquirir conhecimento sobre o [Autenticação.gov](#). Esta autenticação irá permitir que os utilizadores se autenticem na plataforma da [CLAV](#) com recurso ao seu número telemóvel, [PIN](#) da [CMD](#) e [PIN](#) temporário enviado por [SMS](#) ao número de telemóvel. Para um utilizador poder usar esta forma de autenticação necessita de ativar a [CMD](#) no [Autenticação.gov](#) criando um [PIN](#) da [CMD](#).

De seguida, é aprofundado o [Autenticação.gov](#) essencial para permitir esta forma de autenticação.

4.1 ESTADO DA ARTE

4.1.1 *Autenticação.gov*

O [Autenticação.gov](#) surgiu da necessidade de identificação unívoca de um utilizador perante sítios na Web. [2] Será este componente a realizar o processo de autenticação do utilizador e a fornecer os atributos do utilizador necessários para identificar o utilizador numa entidade (*website/portal*).

O [CC](#) em conjunto com o [Autenticação.gov](#) permite obter os identificadores dos utilizadores junto das entidades participantes da iniciativa do [CC](#) (funcionalidade de Federação de Identidades da Plataforma de Interoperabilidade da Administração Pública). Além disso, o [Autenticação.gov](#) gere os vários fornecedores de atributos disponíveis bem como possui uma estreita ligação com a infraestrutura de chave pública do [Cartão de Cidadão \(Public Key Infrastructure \(PKI\)\)](#), com o intuito de manter os elevados níveis de segurança e privacidade no processo de autenticação e identificação. [2]

O [Autenticação.gov](#) permite também a criação de credenciais comuns a todos os sites da [AP](#), ou seja, o utilizador apenas necessita de se autenticar uma vez que poderá aceder aos vários portais (Portal do Cidadão, etc) com a mesma autenticação ([SSO](#)).

Para além disso o utilizador pode autenticar-se utilizando outros certificados digitais que não o [CC](#) (por exemplo [Chave Móvel Digital \(CMD\)](#), *user+password* ou redes sociais, estes dois últimos quando o *website/portal* necessita apenas de conhecer do utilizador o *email*).

No projeto CLAV irá ser implementado a autenticação com recurso ao Autenticação.gov através de dois certificados digitais diferentes:

- **Cartão de Cidadão (CC):** Já se encontra implementado como referido na secção 2.5. A autenticação é realizada através da leitura do CC (através de um leitor de cartões sendo necessário a instalação de *software* do Autenticação.gov para proceder à leitura do CC) e posterior inserção do PIN de autenticação recebido quando se cria/renova o CC;
- **Chave Móvel Digital (CMD):** Um dos objetivos desta tese é a implementação da autenticação com recurso a este certificado digital. Com a CMD, após o utilizador associar um número de telemóvel ao NIC, o utilizador pode autenticar-se com o número de telemóvel, o código PIN da CMD e o código de segurança temporário enviado por SMS.

De forma a completar a figura 3 apresenta-se de seguida, na figura 10, o fluxo de pedidos efetuado entre a CLAV e o Autenticação.gov de forma a autenticar um utilizador na CLAV: [2]

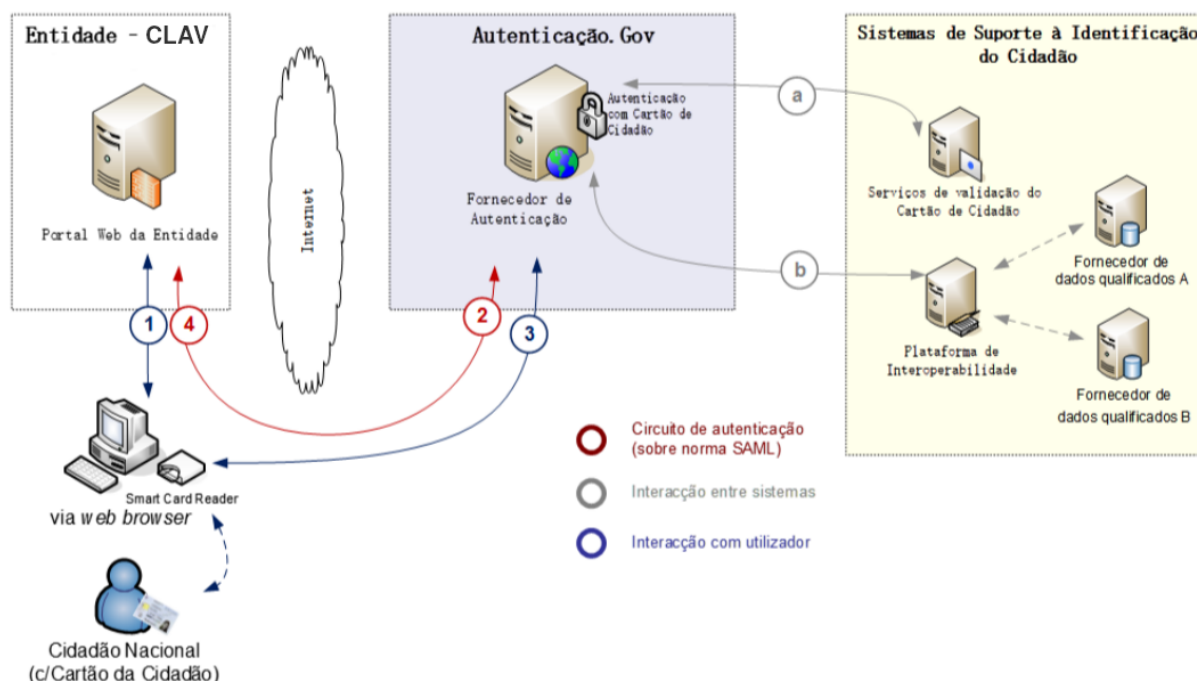


Figura 10: Fluxo de pedidos entre a CLAV e o Autenticação.gov de forma a autenticar um utilizador na CLAV. [2]

1. O utilizador pretende aceder à área privada do portal de uma entidade (da CLAV), na qual é necessário que comprove a sua identidade;
2. O portal da entidade (CLAV) delega a autenticação e redireciona o utilizador para o Autenticação.gov, juntamente com um pedido de autenticação assinado digitalmente;

3. O Autenticação.gov valida o pedido de autenticação recebido e solicita a autenticação do utilizador com recurso ao seu **CC** pedindo a inserção do seu **PIN** de autenticação. Durante este processo, o Autenticação.gov efetua as seguintes operações internas:
 - a) Valida as credenciais do utilizador com recurso à **PKI** do **CC** via **OCSP**;
 - b) Obtém atributos que sejam solicitados pelo portal da entidade (**CLAV**) junto dos vários fornecedores de atributos qualificados. Esta operação é efetuada via Plataforma de Interoperabilidade. Este processo pode incluir a obtenção de dados da Federação de Identidades ou de outras Entidades.
4. A identificação e atributos do utilizador são autenticados e assinados digitalmente pelo Autenticação.gov, após o qual redireciona o utilizador de volta ao portal da entidade original (**CLAV**). Cabe à entidade (**CLAV**) a validação das credenciais do Autenticação.gov e utilização dos atributos do cidadão.

A troca de pedidos entre a **CLAV** e o Autenticação.gov é feita através de **SAML 2.0**.

O **Security Assertion Markup Language (SAML)** define uma *framework standard* em **XML**. [29] Foi aprovado pela **OASIS**, permite a troca segura de informação de autenticação e autorização entre diferentes entidades possibilitando através de uma credencial (*login* de um utilizador) aceder autenticado a um conjunto de *websites* (**SSO**).

Para utilizar o Autenticação.gov é necessário criar um pedido baseado em **SAML**, assinado digitalmente com um certificado X.509 (permite ao Autenticação.gov identificar a entidade responsável pelo pedido) e encriptado com a chave privada de um par de chaves **RSA** (permite ao Autenticação.gov verificar a validade do pedido de autenticação), bem como a respetiva cadeia de autenticação. [35] A chave pública do par de chaves **RSA** faz parte do certificado X.509 que é enviado ao Autenticação.gov.

Há dois tipos de **SAML** usados com o Autenticação.gov [35]:

- **SAML Request (2 da figura 10):** Pedido de autenticação, enviado ao Autenticação.gov, no qual é enviado a origem, assinaturas, atributos a obter do utilizador, etc.
- **SAML Response (4 da figura 10):** Resposta ao pedido de autenticação enviado para o *issuer* do pedido de autenticação. Esta resposta contém o *status* do pedido de autenticação (sucesso, insucesso ou cancelado) e no caso de uma autenticação com sucesso contém os atributos do utilizador requisitados.

Tanto o pedido e a resposta estão no formato **XML** pelo simples facto de ser usado o **SAML 2.0**.

No caso do **SAML Request** o pedido possui o elemento *root* **AuthnRequest** onde se destacam os seguintes atributos:

- **Destination:** URL para o qual o pedido de autenticação é enviado. No caso da Autenticação.gov este apenas pode ser um dos seguintes URL's:
 - <https://preprod.autenticacao.gov.pt/fa/Default.aspx> para o ambiente de teste;
 - <https://autenticacao.gov.pt/fa/Default.aspx> para o ambiente de produção (obrigatório o uso de HTTPS para a comunicação).
- **AssertionConsumerServiceURL:** URL destino da resposta do pedido de autenticação (*SAML Response*). Em produção, é obrigatório que o URL seja em HTTPS;
- **ProviderName:** Nome da plataforma/aplicação que está a requerer a autenticação através do Autenticação.gov. Este nome tem de ser previamente acordado com a AMA durante a emissão da cadeia de autenticação e certificados X.509 [35].

Além destes atributos, o *SAML Request* possui 3 elementos aninhados:

- **Issuer:** Informação sobre quem efetua o pedido de autenticação, ou seja a CLAV. Basta apenas enviar o URL de onde é originário o pedido;
- **Signature:** Assinatura digital do pedido. Aqui é adicionado o certificado X.509 com a cadeia de autenticação fornecida pela AMA;
- **Extensions:** Contém os atributos a requisitar ao Autenticação.gov do utilizador a autenticar-se na CLAV. Aqui são também adicionados o nível de confiança e a política de apresentação do Autenticação.gov:

- **Nível de confiança**

Permite indicar o nível mínimo necessário para realizar a autenticação. Os níveis passíveis de usar são [4]:

- * 4:
 - Autenticação com recurso a uma ou mais operações criptográficas efetuadas no **Cartão de Cidadão** em autenticação que resulta num conjunto de informação fornecida ao Autenticação.gov que permite com o maior grau de certeza de que, no momento da autenticação, se utilizou um **Cartão de Cidadão** real com conhecimento do PIN de autenticação;
 - Autenticação com renegociação SSL com certificado cliente da Ordem dos Notários, da Ordem dos Advogados ou da Câmara dos Solicitadores;
- * 3: Autenticação com **Chave Móvel Digital**;

- * 2: Autenticação com [Chave Móvel Digital](#) através de Email ou *Twitter*;
- * 1: Autenticação com Utilizador/Palavra-passe (também designado por Autenticação Simples) e Redes Sociais.

– Política de apresentação

No Autenticação.gov quando o nível de confiança é inferior a 4 são apresentadas múltiplas abas com os Mecanismos de Autenticação disponíveis. A política de apresentação permite definir o que deve ou não ser apresentado (que abas devem estar disponíveis tendo em conta o nível de confiança definido) e/ou que método de autenticação (aba) deve ser o predefinido. Em caso de conflitos entre o nível de confiança e a política de apresentação ou conflitos na própria política de apresentação, esta é ignorada e é utilizada a política de apresentação que privilegia o CC (mecanismo de autenticação mais seguro). Os Mecanismos de Autenticação disponíveis são [4]:

- * [Cartão de Cidadão](#);
- * [Chave Móvel Digital](#);
- * Utilizador/Palavra-passe também designado por Autenticação Simples;
- * Redes Sociais.

E o mapeamento das abas aos mecanismos de autenticação é [4]:

- * 'CC': Aba relativa à autenticação através de [Cartão de Cidadão](#);
- * 'CMD': Aba relativa à autenticação através de [Chave Móvel Digital](#);
- * 'UPP': Aba relativa à autenticação através de Utilizador/Palavra-passe;
- * 'RSS': Aba relativa à autenticação através das Redes Sociais.

Esta identificação é usada para identificar as abas na extensão em XML.

Após a receção do *SAML Request*, o Autenticação.gov processa a autenticação do utilizador e retorna o *SAML Response* como resposta.

O *SAML Response* tem como elemento *root Response*. Este elemento possui 5 elementos aninhados dos quais se destaca:

- **Status:** Contém a informação relativa ao sucesso ou insucesso do pedido de autenticação;
- **Assertion:** Asserção *SAML* que contém os atributos requisitados sobre o utilizador no caso do pedido de autenticação ter sucesso.

Para finalizar é importante referir que o Autenticação.gov é como se fosse uma caixa negra à qual é enviado um pedido pela entidade que necessita a autenticação do utilizador (neste caso a *CLAV*) com

algumas configurações a usar pelo Autenticação.gov e, esta caixa negra, devolve a informação do utilizador autenticado (os atributos do utilizador requisitados no pedido efetuado ao Autenticação.gov). Os atributos são apenas devolvidos à entidade caso o utilizador autorize o Autenticação.gov a dar à entidade essa informação. A caixa negra trata de autenticar o utilizador através das credenciais fornecidas por este.

Para um maior aprofundamento sobre o Autenticação.gov recomenda-se a leitura da secção 2.2 da dissertação [35] (“CLAV: Autenticação e integração na plataforma iAP” de Octávio Maia).

4.2 SOLUÇÃO

A autenticação com **CMD** é bastante semelhante à autenticação com o **CC**. O que muda é o modo de autenticação perante o Autenticação.gov que em vez de ser efetuado com **CC** (**Cartão de Cidadão**, Leitor de Cartões, *software* local do Autenticação.gov e **PIN** de autenticação do **CC**) será realizado através de **CMD** (Número de telemóvel, **PIN** definido durante a ativação da **CMD** e **PIN** temporário enviado para o número de telemóvel).

Os atributos pedidos ao Autenticação.gov são os mesmos que são pedidos com a autenticação com o **CC** (**NIC** e Nome Completo) permitindo identificar unicamente o utilizador a partir do **NIC**. Para além disso, permite aproveitar todo o back-end e código já desenvolvido para a autenticação do utilizador perante a **CLAV** através de **CC** visto que os atributos usados são os mesmos. Além disso, permite uma melhor normalização da identificação dos utilizadores (não existindo assim vários ids para o mesmo utilizador) bem como evita ser necessário o utilizador partilhar atributos adicionais (como o número de telemóvel) com a **CLAV** (o número de telemóvel é apenas partilhado com o Autenticação.gov sendo que a **CLAV** não tem acesso a esta informação se apenas pediu para saber os atributos **NIC** e Nome Completo). Assim, a única implicação que pode ser retirada daqui é que o utilizador, quer queira usar a autenticação por **CC** quer queira usar por **CMD**, necessita de estar registado na **CLAV** com o **CC** sendo que, neste momento é obrigatório o registo dos utilizadores através de **CC**.

Volta-se a frisar que o **NIC** será usado por forma a identificar o utilizador, pelo que serve de id do utilizador tal como na autenticação com o **CC**.

Portanto, de igual forma como na autenticação com o **CC**, quando é selecionado este método de autenticação, a **CLAV** redireciona o utilizador para o Autenticação.gov que se encarrega de autenticar o utilizador com **CMD**, comunicando, no fim, os atributos pedidos ao Autenticação.gov (**NIC** e Nome Completo). A **CLAV** com o **NIC** verifica se o utilizador já se encontra registado, obtendo mais alguma informação adicional, como o nível de utilizador.

Semelhantemente à autenticação local e à autenticação com o **CC**, é agora gerado um *token* (**JWT**) que irá assinar a partir daqui todos os pedidos do utilizador. O *token* tem a validade de 8 horas, ao fim das quais o utilizador necessita de realizar uma nova autenticação.

Em termos técnicos, é apenas necessário aproveitar o pedido POST que já é enviado ao Autenticação.gov para a autenticação com **CC** e adicionar as extensões nível de confiança e a política de apresentação como é possível constatar em 4.1.1.

A primeira extensão permite indicar o nível mínimo necessário que pretendemos que seja usado pelo Autenticação.gov. Como tal, dado um nível selecionado permite a autenticação através do método desse nível mas também permite os métodos de autenticação dos níveis superiores. Além disso, quanto menor o nível mais fraca é a autenticação, visto que quanto menor o nível mais fraca é a autenticidade das credenciais fornecidas pelo utilizador.

Quando este nível de confiança não está presente (como no caso do pedido enviado para a autenticação com **CC**) o nível de confiança mínimo assumido é o nível máximo (4). Assim para o caso da **CMD** será adicionada a seguinte extensão ao pedido em XML [4]:

```
<fa:FAAALevel xmlns:fa="http://autenticacao.cartaodecidadao.pt/atributos">
  3
</fa:FAAALevel>
```

Exemplo 4.1: Extensão Nível de Confiança no pedido enviado ao Autenticação.gov

Esta extensão indica então que o nível de confiança mínimo necessário é o nível 3, ou seja, permite a autenticação através de **CMD** mas também através de **CC**.

Como tal, por forma a apenas aparecer no Autenticação.gov a hipótese de autenticação através de **CMD** é necessário definir a Política de Apresentação do Autenticação.gov.

Ou seja, é necessário esconder a aba do **Cartão de Cidadão (CC)** e se futuramente se pretender apresentar ambas as abas é também necessário tornar o mecanismo de **Chave Móvel Digital (CMD)** a aba predefinida. Para tal, foi adicionada a seguinte extensão ao pedido em XML [4]:

```
<fa:AuthTabPresentationPolicies xmlns:fa="http://autenticacao.cartaodecidadao.pt/presentationpolicy">
  <!--Torna a aba CMD como predefinida-->
  <fa:defaultSelectedAuthTab TabId="CMD"/>
  <!--Esconde a aba do CC-->
  <fa:hideAuthTab TabId="CC"/>
</fa:AuthTabPresentationPolicies>
```

Exemplo 4.2: Extensão Política de Apresentação no pedido enviado ao Autenticação.gov

Que torna a aba do mecanismo de **CMD** o predefinido e esconde a aba do mecanismo de **CC**.

Portanto, se for enviado um pedido ao Autenticação.gov igual ao da autenticação com **CC** mas com estas extensões, o Autenticação.gov pedirá ao utilizador para se autenticar com **CMD**.

4.3 IMPLEMENTAÇÃO

Nesta secção será demonstrado como se pode utilizar esta forma de autenticação através da interface da CLAV. Para iniciar o processo o utilizador deve aceder a <https://clav.dglab.gov.pt/users/autenticacao> e carregar no botão “Chave Móvel Digital”. Após carregar no botão, o utilizador é redirecionado para o Autenticação.gov onde lhe é apresentada uma página a pedir autorização para a partilha do Nome Completo e do NIC do utilizador com a CLAV como se pode observar na figura 11a. O utilizador ao autorizar, aparece-lhe o formulário para a inserção do número de telemóvel e do PIN da CMD como presente na figura 11b. Após carregar em “Autenticar” se os valores estiverem corretos aparecerá o formulário para introduzir o PIN temporário enviado por SMS como se observa na figura 11c.



Figura 11: Autenticação.gov: Processo de autenticação com CMD

Ao fim de carregar em “Confirmar” e caso o PIN temporário esteja correto, o utilizador é redirecionado de volta para a CLAV onde aparecerá ao utilizador uma de duas hipóteses:

- Se o utilizador já se encontra registado na CLAV então autentica-se com sucesso onde é redirecionado para a *Home Page* da CLAV como se observa na figura 12:

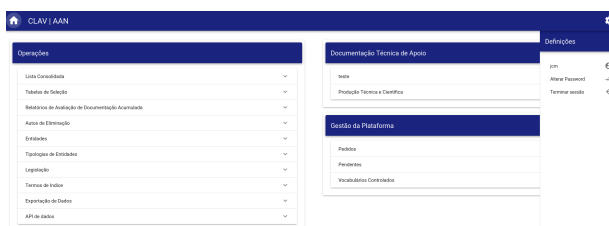


Figura 12: Utilizador autenticado com sucesso através de CMD

- Se o utilizador ainda não se encontra registado na [CLAV](#) então não é autenticado na [CLAV](#) apresentando uma mensagem de erro como na figura 13:



Figura 13: Mensagem de erro na [CLAV](#) caso não se encontre registado na [CLAV](#) mas tenha se autenticado com sucesso no [Autenticação.gov](#)

Por fim, caso o utilizador cancele algum dos passos de autenticação do [Autenticação.gov](#), se engane em alguma credencial ou aconteça algum erro no [Autenticação.gov](#) é redirecionado de volta para a [CLAV](#) onde lhe é apresentado a mensagem de erro presente na figura 14:



Figura 14: Mensagem de erro na [CLAV](#) após falha no [Autenticação.gov](#)

4.4 RESUMO

Após um breve aprofundamento do [Autenticação.gov](#) foi descrita a solução desenvolvida e na secção da implementação apresentou-se o processo de autenticação através de [Chave Móvel Digital](#) presente.

DOCUMENTAÇÃO DA API DA CLAV

A documentação da [API](#) de dados é bastante importante por forma a ajudar futuros programadores e utilizadores a usarem a plataforma da [CLAV](#).

Para proceder à documentação em *Swagger* da [API](#) de dados da [CLAV](#) é essencial perceber o que é o *Swagger* (*OpenAPI* e *Swagger UI*) e suas alternativas. Isto é observado no estado da arte que se apresenta a seguir.

5.1 ESTADO DA ARTE

5.1.1 *Swagger*

O *Swagger* é um ecossistema de ferramentas para desenvolver [APIs](#) com a [OpenAPI Specification \(OAS\)](#). Até 2015 o *Swagger* consistia numa especificação e num ecossistema de ferramentas para implementar a especificação. Em 2015 a fundadora do *Swagger*, *SmartBear Software*, doou a especificação *Swagger* para a *Linux Foundation* e renomeou a especificação para [OpenAPI Specification](#). [50]

A especificação *OpenAPI* é agora desenvolvida pela *OpenAPI Initiative* que envolve várias empresas tecnológicas entre as quais *Microsoft*, *Google*, *IBM* e a fundadora *Smartbear Software*.

Já o conjunto de ferramentas *Swagger* inclui ferramentas *open-source*, gratuitas e comerciais que podem ser usadas em diferentes estágios do ciclo de vida de uma [API](#), que inclui documentação, desenho, testes e *deployment*. Algumas das ferramentas são: [46]

- ***Swagger Editor***. Permite editar especificações *OpenAPI* em [YAML](#) no *browser*¹, validar as especificações em relação às regras do [OAS](#) bem como pré-visualizar a documentação em tempo real. Facilita o desenho e a documentação de [APIs REST](#);
- ***Swagger UI***. Coleção de *assets HTML*, *JavaScript* e *CSS* que geram dinamicamente documentação a partir de uma especificação *OpenAPI* de uma [API](#);

¹Aceder <https://editor.swagger.io/>

- **Swagger Codegen:** Permite a geração de bibliotecas cliente (geração de [SDK](#)), *server stubs* e documentação automática a partir de especificações *OpenAPI*;
- **Swagger Inspector:** Ferramenta de testes de [APIs](#) que permite validar as [APIs](#) e gerar definições *OpenAPI* de [APIs](#) existentes;
- **SwaggerHub:** Desenho e documentação de [APIs](#), construído para equipas que trabalham com *OpenAPI*.

O *Swagger* possui duas abordagens: [56]

- *top-down:* Uso do *Swagger Editor* para criar a especificação *OpenAPI* e depois usar o *Swagger Codegen* por forma a gerar o código do cliente e do servidor. Ou seja, primeiro desenha-se a [API](#) antes de escrever código;
- *bottom-up:* Utilizador já possui uma [API REST](#) e o *Swagger* irá ser usado apenas para documentar a [API](#) existente.

Visto que a [CLAV](#) já possui grande parte da [API](#) construída vai ser usada uma abordagem *bottom-up*. Portanto, o *Swagger* vai ser usado apenas para a documentação da [API](#). De forma a produzir a documentação, do portfólio de ferramentas do *Swagger* apenas precisaremos de utilizar o *Swagger UI* e o *Swagger Editor*. O primeiro permitirá apresentar aos utilizadores a documentação gerada e o segundo permitirá validar a especificação *OpenAPI* (documentação) criada, verificando se não possui erros.

5.1.2 Especificação *OpenAPI*

A especificação *OpenAPI* providencia um conjunto de propriedades que podem ser usadas para descrever uma [API REST](#). Com um documento de especificação válido é possível usá-lo para criar uma documentação interativa, por exemplo, através do *Swagger UI*.

De seguida será apresentado o que é possível documentar com a especificação *OpenAPI* e como. É possível usar tanto [YAML](#) como [JSON](#) para a especificar. Esta parte será demonstrada usando [YAML](#).²

Metadata

O primeiro passo é escolher a versão da especificação *OpenAPI* que irá ser usada para documentar:

```
openapi: 3.0.0
```

Exemplo 5.1: Exemplo de indicação da versão da especificação *OpenAPI*

²A especificação completa do *OpenAPI* com versão igual a 3.0.0 pode ser vista em <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md>

Depois, na secção `info`, é possível descrever um pouco a [API](#) que estamos a documentar, indicando o título, a descrição e a versão da mesma. As propriedades `title` e `version` são obrigatórias. É possível também colocar informação sobre os contactos disponíveis, termos de uso e a licença:³

```
info:
  title: CLAV API
  description: Esta é a API do projeto CLAV...
  version: 1.0.0
```

Exemplo 5.2: Exemplo de secção `info` indicando título, descrição e versão da [API](#) na especificação *OpenAPI*

Servidores

Há depois uma secção com o nome de `servers` para indicar os *URLs* que são os pontos de acesso da [API](#). Podem ser indicados vários pontos de acesso:⁴

```
servers:
- url: http://clav-api.dglab.gov.pt/api
  description: Official API server
- url: http://clav-test.di.uminho.pt/api
  description: Testing server
- url: http://localhost:7779/api
  description: Local server
```

Exemplo 5.3: Exemplo de secção `servers` indicando os *URLs* e a descrição de cada na especificação *OpenAPI*

Caminhos/Rotas

De seguida apresenta-se uma das secções mais importantes da especificação, a secção `paths`. Aqui são definidas as rotas que a [API](#) disponibiliza. Para definir cada rota basta indicar o caminho relativo aos pontos de acesso definidos na secção `servers` (`<server-url>/<caminho relativo>`). Nesta secção é definido tudo o que envolve as rotas, desde os parâmetros necessários, as respostas que devolve, os métodos *HTTP* disponíveis, etc:^{5,6}

```
paths:
  /users/{id}:
    get:
      summary: Resumo do que faz a rota
      description: >
        Descrição detalhada, pode ser usado Markdown para enriquecer o texto
      parameters:
        - name: id
          in: path
          description: Id do utilizador
```

³Ver mais em <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#infoObject>

⁴Para mais detalhes sobre esta secção veja <https://swagger.io/docs/specification/api-host-and-base-path/>

⁵Mais detalhes em <https://swagger.io/docs/specification/paths-and-operations/>

⁶mais detalhes sobre a funcionalidade `$ref` em <https://swagger.io/docs/specification/using-ref/>

```

    required: true
    schema:
      type: string
  responses:
    200:
      description: Descrição da resposta, p.e: Sucesso
      content:
        application/json:
          schema:
            #A estrutura do JSON devolvido pode ser definido logo aqui ou num componente à parte, fazendo
            #referência desse. Iremos aplicar o segundo caso para demonstrar que estas funcionalidades
            #tornam a documentação mais fácil de manter
            $ref: '#/components/schemas/User'
    ...
  post:
    ...
  delete:
    ...
/users:
  ...

components:
  schemas:
    User:
      type: object
      properties:
        id:
          type: string
        ...
      required:
        - id
        ...

```

Exemplo 5.4: Exemplo de secção paths indicando os detalhes de cada rota na especificação *OpenAPI*

Outro ponto importante a referir é que é possível agrupar as rotas em grupos através do uso de *tags*. As *tags* têm de ser definidas numa secção chamada *tags*:

```

tags:
- name: users
  description: Descrição
- name: classes
  description: Outra descrição

```

Exemplo 5.5: Exemplo de secção tags definindo tags na especificação *OpenAPI*

Depois em cada rota é necessário indicar a que *tag* (grupo) pertence:

```

paths:
  /users/{id}:
    get:

```

```
summary: Resumo do que faz a rota
tags:
  - users
...
```

Exemplo 5.6: Exemplo de uso de *tags* numa rota na especificação *OpenAPI*

Parâmetros

Como já exemplificado no exemplo 5.4 os parâmetros de uma rota são definidos na secção `parameters` de cada rota. Existem quatro tipos de parâmetros que variam de acordo com o local onde se encontram. O tipo de um parâmetro é definido na propriedade `in` de um parâmetro e pode ser um dos seguintes:

- Parâmetros no caminho: Servem normalmente para apontar para um recurso específico. Estes parâmetros são sempre obrigatórios como tal a propriedade `required` com o valor igual a verdadeiro deve ser sempre adicionada. Para além disso o `name` tem de ser igual ao que está no caminho. A propriedade `in` tem o valor de *path*;
- Parâmetros na *query string*: A propriedade `in` tem o valor de *query*. No caso de tokens passados em parâmetros da *query string* devem-se usar esquemas de segurança, veja a secção 5.1.2;
- Parâmetros no cabeçalho: A propriedade `in` tem o valor de *header*. Contudo os cabeçalhos *Accept*, *Content-Type* e *Authorization* não são aqui definidos;
- Parâmetros no cabeçalho da *Cookie*: A propriedade `in` tem o valor de *cookie*.

Cada parâmetro tem várias propriedades que permitem defini-lo:⁷

- `required`: Indica se o parâmetro é obrigatório ou opcional. Possíveis valores são *true* ou *false*.
- `schema`:
 - `default`: Valor padrão de um parâmetro opcional;
 - `type`: O tipo do parâmetro. Possíveis valores: *string*, *integer*, etc;
 - `enum`: Indica os possíveis valores para o parâmetro;
 - `nullable`: Indica se o parâmetro pode ser *null*. Possíveis valores são *true* ou *false*.
- `allowEmptyValue`: Indica se o parâmetro pode ser vazio. Apenas aplicável no caso de um parâmetro na *query string*. Possíveis valores são *true* ou *false*;
- `example`: Um exemplo do valor;

⁷Mais detalhes em <https://swagger.io/docs/specification/describing-parameters/>

- `examples`: Múltiplos exemplos;
- `deprecated`: Indica se o parâmetro é ou não *deprecated*. Possíveis valores são *true* ou *false*.

Request Body

O *request body* é definido em cada rota na secção `requestBody` sendo usado essencialmente em rotas com o método `HTTP` igual a `POST` ou a `PUT`, ou seja, em casos que há necessidade de criar ou alterar um objeto de acordo com a informação fornecida no pedido. As propriedades que podem ser definidas no `requestBody` são as seguintes:^{8,9}

- `description`: Opcionalmente pode ser adicionada uma descrição;
- `required`: Indica se o *request body* é obrigatório ou opcional. Possíveis valores são *true* ou *false*. Por omissão o *request body* é opcional;
- `content`: Obrigatório. Lista os *media types* consumidos pela rota e especifica o `schema` para cada *media type*.

Respostas

Nesta secção, propriedade `responses` de cada rota, é descrita as possíveis respostas de cada rota. Na propriedade serão definidas as várias respostas, uma resposta por cada `HTTP status code` possível de ser devolvido pela rota. Cada resposta pode possuir as seguintes propriedades:¹⁰

- `description`: Obrigatório, descrição da resposta;
- `content`: Opcional, semelhante ao `content` do *request body* e define o conteúdo que é devolvido;
- `headers`: Opcional, define as *headers* que são devolvidas na resposta.

Adição de Exemplos

Na secção 5.1.2 já se referiu como é possível adicionar exemplos aos parâmetros. De forma semelhante o mesmo pode ser realizado tanto no *request body* como nas respostas através da propriedade `example` (um exemplo) ou `examples` (múltiplos exemplos) aninhado(s) na propriedade `schema` ou aninhado(s) na “propriedade” *media type* no caso do `schema` ser uma referência para um modelo presente na secção

⁸Para mais detalhes, desde *upload* de ficheiros, a *Form Datas*, veja em <https://swagger.io/docs/specification/describing-request-body/>

⁹Para mais informação sobre os *media types* veja <https://swagger.io/docs/specification/media-types/>

¹⁰Mais detalhes em <https://swagger.io/docs/specification/describing-responses/>

components. A propriedade `example` pode também ser usada em objetos ou propriedades de um schema. Por fim, para adicionar exemplos de XML ou HTML os exemplos devem ser *strings*:¹¹

```
content:
  application/xml:
    schema:
      $ref: '#/components/schemas/xml'
    examples:
      xml:
        summary: A sample XML response
        value: '<objects><object><id>1</id><name>new</name></object><object><id>2</id></object></objects>'
  text/html:
    schema:
      type: string
    examples:
      html:
        summary: A list containing two items
        value: '<html><body><ul><li>item 1</li><li>item 2</li></ul></body></html>'
```

Exemplo 5.7: Exemplo de adição de exemplos para XML e HTML na especificação *OpenAPI*

Modelos

A secção `schemas` presente na secção `components` permite definir estruturas de dados (modelo) a serem usados na API. Estes modelos podem ser referenciados usando a funcionalidade `$ref`.¹²

Autenticação e Autorização

Nesta secção será demonstrada como se pode adicionar a autenticação e autorização à especificação *OpenAPI*. Para tal é necessário criar *security schemes*. Os esquemas são definidos na secção `securitySchemes` dentro da secção `components`. Para cada esquema de segurança é necessário definir a propriedade `type`. Na especificação é possível descrever os seguintes esquemas de segurança:

- Esquemas de autenticação HTTP (usam o cabeçalho *Authorization*) (`type = http`):
 - *Basic* (propriedade `scheme = basic`);
 - *Bearer* (propriedade `scheme = bearer`);
 - Outros esquemas HTTP definidos pelo RFC 7235 e pelo registo de esquemas de autenticação HTTP.
- Chaves API no cabeçalho, na *query string* ou em *cookies* (`type = apiKey` e na propriedade `in` indicar em que local se encontra, se no cabeçalho (`header`), se na *query string* (`query`) ou se nas *cookies* (`cookie`));

¹¹Mais detalhes em <https://swagger.io/docs/specification/adding-examples/>

¹²Mais detalhes em <https://swagger.io/docs/specification/data-models/>

- *OAuth 2* (type = oauth2);
- *OpenID Connect Discovery* (type = openIdConnect).

Após definir os esquemas de segurança é necessário aplicá-los nas rotas que devem estar protegidas por esses esquemas. Para tal em cada rota pode ser definida a propriedade `security` e indicar os esquemas de segurança que essa rota suporta.¹³

Alternativas

Em termos de alternativas à especificação *OpenAPI* existem duas concorrentes: *RAML*¹⁴ e *API Blueprint*¹⁵. Comparemos as três hipóteses: [47]

Especificação	Vantagens	Desvantagens
<i>OpenAPI</i>	<ul style="list-style-type: none"> • Grande adoção • Grande comunidade de utilizadores • Bom suporte • Suporte para várias linguagens 	<ul style="list-style-type: none"> • Falta de construtores avançados para metadados
<i>RAML</i>	<ul style="list-style-type: none"> • Suporta construções avançadas • Adoção decente • <i>Human readable format</i> • Grande apoio da indústria 	<ul style="list-style-type: none"> • Falta de ferramentas ao nível do código • Ainda não comprovado a longo prazo
<i>API Blueprint</i>	<ul style="list-style-type: none"> • Fácil de entender • Simples de escrever 	<ul style="list-style-type: none"> • Pouca adoção • Falta de construtores avançados • Instalação complexa

Tabela 1: Comparação entre especificações de documentação de APIs

Além das vantagens apresentadas as três são *open-source*. Para além disso, o *OpenAPI* pode ser escrito em *JSON* ou *YAML*, o *RAML* é escrito em *YAML* e o *API Blueprint* é escrito em *Markdown*. Escolheu-se a especificação *OpenAPI* devido à sua grande adoção e por permitir usar o ecossistema de ferramentas *Swagger*.

¹³Mais detalhes em <https://swagger.io/docs/specification/authentication/>

¹⁴Ver <https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/>

¹⁵Ver <https://github.com/apiaryio/api-blueprint/blob/master/API%20Blueprint%20Specification.md>

5.1.3 Swagger UI

O *Swagger UI* permite a qualquer um visualizar uma **API REST**. A partir de um documento **JSON** ou **YAML** (especificação *OpenAPI*) é automaticamente gerado uma documentação interativa. Na figura 15 está presente a visualização da documentação de uma **API** através do *Swagger UI*.

The screenshot displays the Swagger UI for the CLAV API. At the top, there's a header with the Swagger logo and the API name 'CLAV API 2.0.0 OAS3'. Below this, there's a section for 'Auth' with three endpoints: GET /auth/{id}, POST /auth/adicionar, and DELETE /auth. Below that, there's a section for 'Autos de Eliminação' with two endpoints: GET /autosEliminacao and GET /autosEliminacao/{id}. The interface also shows server variables and an 'Authorize' button.

Figura 15: *Swagger UI* exemplo

Alternativas

Existem várias alternativas ao *Swagger UI*:

Ferramenta	Vantagens	Desvantagens
<i>Swagger UI</i>	<ul style="list-style-type: none"> • Suporta a especificação <i>OpenAPI</i> • <i>Open-source</i> • Amplamente usado 	
<i>Apiary</i> ¹⁶	<ul style="list-style-type: none"> • Suporta a especificação <i>API Blueprint</i> e a especificação <i>OpenAPI</i> 	<ul style="list-style-type: none"> • Necessário pagar de forma a poder integrar a documentação da <i>API</i> num domínio próprio • <i>Closed-source</i>
<i>API Console</i> ¹⁷	<ul style="list-style-type: none"> • Suporta a especificação <i>RAML</i> e a especificação <i>OpenAPI</i> • <i>Open-source</i> 	
<i>Slate</i> ¹⁸	<ul style="list-style-type: none"> • <i>Open-source</i> • <i>API</i> definida em <i>Markdown</i> 	<ul style="list-style-type: none"> • Não suporta nenhuma especificação
<i>apiDoc</i> ¹⁹	<ul style="list-style-type: none"> • Documentação criada a partir das anotações nos comentários do código • <i>Open-source</i> 	<ul style="list-style-type: none"> • Não suporta nenhuma especificação
<i>ReDoc</i> ²⁰	<ul style="list-style-type: none"> • Suporta a especificação <i>OpenAPI</i> • <i>Open-source</i> • Fácil de integrar 	

Tabela 2: Comparação entre ferramentas de APIs

5.1.4 Produção da documentação da API da CLAV

Neste secção são aprofundadas algumas bibliotecas que podem ser usadas para a produção da documentação com a especificação *OpenAPI* e o *Swagger UI*.

De seguida, são aprofundadas duas *packages* que podem ser usadas para criar documentação interativa (integração do *Swagger UI*) para uma *API REST* criada com *Node.js* e *Express.js*: [56]

- `swagger-node-express`

¹⁶Ver <https://apiary.io/>

¹⁷Ver <https://github.com/mulesoft/api-console>

¹⁸Ver <https://github.com/slatedocs/slate>

¹⁹Ver <https://apidocjs.com/>

²⁰Ver <https://github.com/Redocly/redoc>

– Vantagens

- * Módulo oficial suportado pelo *Swagger*;
- * É *open-source* e como tal é possível contribuir para a correção de problemas;
- * A solução contém *Swagger Editor* e *Swagger Codegen* e como tal tanto podemos usar uma abordagem *top-down* como *bottom-up*.

– Desvantagens

- * Instalação manual do *Swagger UI*. O código do *Swagger UI* tem de ser copiado manualmente para o projeto e sempre que há uma atualização é necessário copiar novamente manualmente;
- * Instalação complexa. Por forma a aplicação hospedar a documentação é necessário adicionar algumas rotas ao servidor para além das já definidas na especificação *OpenAPI*;
- * Fraca documentação.

• `swagger-ui-express`

– Vantagens

- * É *open-source* e como tal é possível contribuir para a correção de problemas;
- * Não é necessário copiar manualmente o *Swagger UI*;
- * De fácil instalação, apenas é necessário adicionar uma rota aonde estará hospedada a documentação;
- * Boa documentação.

– Desvantagens

- * Não é o módulo oficial suportado pelo *Swagger*.

Das duas, a `swagger-ui-express` é a de mais simples implementação e de mais fácil manutenção.

```
var swaggerUI = require('swagger-ui-express')
//JSON
var swaggerDocument = require('./swagger.json')
//ou YAML
var yaml = require('js-yaml')
var fs = require('fs')
var swaggerDocument = yaml.load(fs.readFileSync('./swagger.yaml'))

app.use('/doc', swaggerUI.serve, swaggerUI.setup(swaggerDocument));
```

Exemplo 5.8: Exemplo de uso do `swagger-ui-express`

No exemplo 5.8 a documentação da API está presente na rota `/doc`. Neste exemplo percebe-se como carregar uma especificação *OpenAPI* em JSON bem como em YAML. Quanto ao *middleware* `serve` retorna os ficheiros estáticos necessários para hospedar o *Swagger UI*. Já o segundo *middleware* `setup` para além de poder receber o documento com a especificação *OpenAPI* pode também receber um outro parâmetro de opções que o utilizador pode definir para a apresentação interativa da documentação com o *Swagger UI*²¹.

Agora há duas abordagens possíveis para realizar a documentação:

- Documentação de cada rota nos comentários da rota através da utilização da *package* `swagger-jsdoc`²²;
- Documentação separada do código.

A abordagem que mantém a documentação separada do código permite modularizar a documentação. A modularização da documentação pode ser realizada através do uso da *package* `yaml-include`²³. Esta *package* permite que o documento YAML da especificação *OpenAPI* possa ser dividida por vários ficheiros. A *package* permite a inclusão de arquivos YAML externos ou a inclusão de pastas de ficheiros YAML. Esta funcionalidade é desaprovada pela equipa de desenvolvimento do YAML contudo ajuda e simplifica a construção do ficheiro de especificação *OpenAPI*.

```
openapi: 3.0.0
info:
  description: Esta é a API do projeto CLAV. Pode encontrar mais informação sobre o CLAV em [http://clav.dglab.gov.pt](http://clav.dglab.gov.pt).
  version: 1.0.0
  title: CLAV API
  contact:
    name: CLAV
    email: clav@dglab.gov.pt
servers:
  - url: http://localhost:7779/api
    description: Local API server
paths: !!inc/dir [ 'paths' ]
components:
  schemas: !!inc/dir [ 'schemas', excludeTopLevelDirSeparator: true ]
  securitySchemes: !!inc/file '/security/schemes.yaml'
```

Exemplo 5.9: Exemplo de uso do `yaml-include` no documento de especificação *OpenAPI*(*index.yaml*)

O ficheiro *index.yaml* será a raiz do documento de especificação *OpenAPI* a ser gerado com a *package* `yaml-include`. A seguinte estrutura dos ficheiros exemplifica como se pode dividir a documentação por vários ficheiros com esta *package* para gerar o documento de especificação *OpenAPI*:

²¹As opções possíveis estão presentes em <https://github.com/scottie1984/swagger-ui-express>. Para o atributo (opção) `swaggerOptions` as opções possíveis estão presentes em <https://github.com/swagger-api/swagger-ui/blob/master/docs/usage/configuration.md>

²²Ver <https://github.com/Surnet/swagger-jsdoc>

²³Ver <https://github.com/claylo/yaml-include>

```

* index.yaml
* paths/
  * classes/
    * get.yaml
    * ~id/
      * get.yaml
  * users/
    * ~id/
      * post.yaml
      * delete.yaml
* schemas/
  * User.yaml
* security/
  * schemes.yaml

```

Exemplo 5.10: Exemplo de estrutura dos ficheiros para gerar o documento de especificação *OpenAPI*

Assim, o `!!inc/dir` fará com que no ficheiro *index.yaml* na *tag paths* sejam incluídos todos os ficheiros que estão na pasta *paths*. Cada ficheiro corresponderá a uma determinada rota com um determinado método **HTTP**. O método **HTTP** é definido a partir do nome do ficheiro e o caminho da rota é determinado pelo nome das pastas e do aninhamento destas. Quando o nome da pasta é iniciado por “~” no caminho será colocado o nome da pasta sem o til e entre chavetas (“{}”) por forma a indicar um parâmetro que é colocado no caminho do pedido.

Já no caso do `!!inc/dir` dos *schemas* a opção `excludeTopLevelDirSeparator` permite que os ficheiros que estejam dentro da pasta *schemas* (mas não aninhados dentro de outras pastas) sejam incluídos sem qualquer aninhamento, assumindo o nome do ficheiro como o atributo a colocar.

Existe também o `!!inc/file` que permite incluir sobre uma determinada *tag* a informação presente no ficheiro referenciado pelo caminho.

O documento de especificação *OpenAPI* final gerado será:

```

openapi: 3.0.0
info:
  description: Esta é a API do projeto CLAV. Pode encontrar mais informação sobre o CLAV em [http://clav.dglab.gov.pt](http://clav.dglab.gov.pt).
  version: 1.0.0
  title: CLAV API
  contact:
    name: CLAV
    email: clav@dglab.gov.pt
servers:
  - url: http://localhost:7779/api
    description: Local API server
paths:
  /classes:
    get:
      <conteúdo do ficheiro paths/classes/get.yaml>

```



```

/classes/{id}:
  get:
    <conteúdo do ficheiro paths/classes/~id/get.yaml>
/users/{id}:
  post:
    <conteúdo do ficheiro paths/users/~id/post.yaml>
  delete:
    <conteúdo do ficheiro paths/users/~id/delete.yaml>
components:
  schemas:
    User:
      <conteúdo do ficheiro schemas/User.yaml>
  securitySchemes:
    <conteúdo do ficheiro security/schemes.yaml>

```

Exemplo 5.11: Documento de especificação *OpenAPI* gerado a partir do ficheiro *index.yaml* com o uso da *package* `yaml-include`

No final teremos um ficheiro no formato [YAML](#) com toda a documentação da [API](#) que poderá então ser usado para alimentar a documentação dinâmica *Swagger UI*.

5.2 SOLUÇÃO

No estado da arte foram abordadas várias alternativas ao *Swagger UI*. A partir da tabela 2 e tendo em conta que:

- Não há financiamento;
- Já existe uma [API](#) desenvolvida;
- A documentação deve estar acessível de um domínio próprio;
- A documentação deve ser fácil de criar, de editar e de manter;
- Será usada a especificação *OpenAPI*.

As várias alternativas ficam reduzidas ao *Swagger UI* e ao *ReDoc*. Optou-se por escolher o *Swagger UI* visto ser a ferramenta mais amplamente usada para além de que é possível obter também uma fácil integração no *Swagger UI* com recurso à *package* `swagger-ui-express` já aprofundada no estado da arte (ver 5.1.4). Além disso, escolheu-se a *package* `yaml-include` por forma a auxiliar a produção da documentação na criação do ficheiro com a especificação *OpenAPI*, permitindo que esta documentação seja modular.

A documentação modular será estruturada da seguinte forma:

```
* /index.yaml
* /paths/
  * /classes/
    * /get.yaml
    * /post.yaml
    * /~id/
      * /get.yaml
      ...
    ...
  ...
* /examples/
  * /classes/
    * /ClasseCompletaJSON.yaml
    * /ClasseCompletaXML.yaml
    * /ClasseSimplesJSON.yaml
    ...
  ...
* /schemas/
  * /classes/
    * /ClasseCompleta.yaml
    ...
  * /definicoes/
    * Codigo.yaml
    ...
  ...
```

Exemplo 5.12: Excerto da estrutura modular da documentação

Portanto nesta estrutura na pasta `paths` é seguida a estrutura do `yaml - include` já descrita no estado da arte em 5.10 em que as pastas indicam o caminho e o nome dos ficheiros o verbo `HTTP` da rota. Em cada um destes ficheiros é colocada a descrição, os parâmetros, etc, da rota. Nestes ficheiros serão feitas referências aos `schemas` e `examples` necessários, estando estes nas pastas `schemas` e `examples` respetivamente. Isto permite que os modelos e os exemplos sejam reutilizados para além de que permite tornar os ficheiros onde são referenciados mais fáceis de perceber, principalmente nos casos em que os modelos e os exemplos são extensos.

Em termos de organização das pastas `schemas` e `examples`, estas no primeiro “nível” possuem pastas de cada grupo de rotas (`classes`, `entidades`, `users`, `chaves`, etc) e em cada uma destas pastas estão os modelos/exemplos correspondentes a esse grupo de rotas. Na pasta `schemas` existe contudo uma pasta especial chamada `definicoes` que possui os vários tipos de dados, como por exemplo o código de uma classe, o id de uma entidade, etc. Assim, estes modelos são usados nos outros modelos e nas rotas permitindo uniformizar estes dados. Ou seja, caso haja por exemplo alguma alteração no formato do código, será apenas necessário alterar no modelo presente na pasta `definicoes` que a alteração será “propagada” já que todos que precisam deste modelo incluem-no por referência.

Portanto, sempre que é criado um novo modelo ou uma nova rota, convém verificar nesta pasta `definicoes` se já existe o tipo de dados necessário a usar bastando assim fazer-lhe referência.

Por fim, no ficheiro `index.yaml` são definidos os grupos de rotas (*tags*) bem como várias informações gerais sobre a API de dados, como descrição, métodos de autenticação, etc. Além disso, é aqui também definido, apenas no caso da pasta `examples`, que ficheiros deve o `yaml-include` ignorar ao incluir a pasta `examples` sobre a *tag* `examples` dos `components`. Isto é efetuado por forma a evitar erros de sintaxe da especificação *OpenAPI*. Estes erros acontecem porque há vários ficheiros de exemplos a serem incluídos nos ficheiros das rotas diretamente pelo `yaml-include` em vez de se usar o `$ref` da especificação *OpenAPI* em casos que o `$ref` não permite efetuar o pretendido. Assim, estes ficheiros têm de ser ignorados.

Com o `yaml-include`, sempre que a API de dados inicia é criado o ficheiro de especificação *OpenAPI* final que incluirá os dados destes ficheiros nos seus locais apropriados. Este ficheiro final ficará disponível na pasta pública do servidor com o nome `clav.yaml` ou seja acessível a partir de <https://clav-api.dglab.gov.pt/clav.yaml>. É este o ficheiro que alimenta o *SwaggerUI* construído com o *swagger-ui-express* e disponível em <https://clav-api.dglab.gov.pt/v2/docs>.

5.3 IMPLEMENTAÇÃO

A documentação da API de dados da CLAV já se encontra disponível em <https://clav-api.dglab.gov.pt/v2/docs>, possuindo uma descrição inicial de como os utilizadores podem obter *tokens* e usar a API de dados a partir do *Swagger UI*. Esta documentação em cada rota possui uma descrição, *query strings* que podem ser definidas, exemplos de *bodies* quando aplicável, possíveis respostas bem como exemplos de respostas. Além disso, tanto os *bodies* e as respostas quando possuem exemplos possuem também o esquema desse *body/resposta*.

Para experimentar uma rota, é então apenas necessário selecionar uma rota, carregar em *Try it out*, inserir os valores que pretende e carregar em *Execute*. Isto claro sem a autenticação. Com autenticação, após obter o *token* para o usar deve carregar no cadeado aberto, selecionar a forma de colocar o *token* no pedido que pretende:

- para Chaves API:
 - Na *query string*: `apiKeyQuery`
 - No cabeçalho `Authorization`: `apiKeyAuth`
- para Utilizadores:
 - Na *query string*: `userQuery`
 - No cabeçalho `Authorization`: `userAuth`

inserir o *token* corretamente (atenção que os casos em que é inserido no cabeçalho *Authorization* são especiais), carregar em *Authorize* e depois já poderá aceder às rotas. Ao efetuar este passo uma vez, o *Swagger UI* usa o mesmo *token* nas restantes rotas que possui a mesma forma de autenticação não necessitando deste passo nessas (aquelas em que o cadeado se encontra fechado).

5.4 RESUMO

Resumidamente, após entender a especificação *OpenAPI*, o *Swagger UI* e possível abordagem de escrita no estado da arte deste capítulo, descreveu-se na secção solução como a documentação é escrita e apresentada ao utilizador.

Por fim, na secção da implementação descreveu-se de que forma pode ser usada esta documentação já disponível em <https://clav-api.dglab.gov.pt/v2/docs>.

EXPORTAÇÃO DE DADOS

Esta tese tem como um dos objetivos a exportação de dados da [API de Classes \(LC\)](#), Entidades, Tipologias e Legislações em formato [JSON](#), [XML](#) e [CSV](#). Além disso, deve permitir a exportação da ontologia (possuidor da informação das variantes referidas a exportar) nos formatos [RDF](#) seguintes: [Turtle](#), [JSON-LD](#) e [RDF/XML](#).

A [API](#) de dados já devolve a sua informação em [JSON](#). Portanto, por forma a realizar a exportação dos dados para outros formatos é necessário realizar a conversão de [JSON](#) para o formato de saída pretendido.

Como tal, este capítulo inicia, no estado da arte, com a investigação de algumas bibliotecas disponíveis no [NPM](#) que têm como objetivo exportar de [JSON](#) para [XML](#) ou de [JSON](#) para [CSV](#).

6.1 ESTADO DA ARTE

6.1.1 XML

Começemos por perceber o que é o [Extensible Markup Language \(XML\)](#). Como se pode perceber pelo nome o [XML](#) é uma linguagem *Markup*, ou seja, uma linguagem que anota o texto para que a máquina possa manipular o texto de acordo com as anotações. O [XML](#) foi desenhado para ser de fácil leitura tanto para humanos como para máquinas e tem como principal intuito o armazenamento e transporte de informação (o tal texto anotado). Além disso é extensível visto que permite que criemos as nossas *tags*.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<peessoa>
  <nome>Maria</nome>
  <idade tipo="numero">23</idade>
  <morada>
    <pais>Portugal</pais>
    <cidade>Braga</cidade>
  </morada>
  <altura unidade="cm">160</altura>
  <!--Isto é um comentário-->
</peessoa>
```

Exemplo 6.1: Pequeno exemplo em [XML](#)

De seguida serão apresentadas, de forma simplificada, as regras de sintaxe aplicadas ao XML para cada componente deste.

- **Declaração XML**

```
<?xml version = "1.0" encoding = "UTF-8"?>
```

Opcional, especifica a versão do XML e o *encoding* usado no documento

- A declaração é *case-sensitive* pelo que deve começar obrigatoriamente por xml
- Se presente no documento, a declaração tem de ser obrigatoriamente o primeiro componente

- **Tags e elementos**

```
<nome>Maria</nome> ou <semnome/>
```

- Cada elemento tem de ser fechado seja por uma *tag* (*tag* final), ou como apresentado acima pela própria *tag*.
- Os elementos XML podem conter elementos filhos mas estes não se podem sobrepor, ou seja, uma *tag* final de um elemento tem de ter o mesmo nome que a última *tag* inicial ainda sem *tag* final.
- Um documento XML apenas pode ter um elemento na raiz do documento (elemento *root*)
- Os nomes das *tags* são *case-sensitive*, portanto, a *tag* inicial e final de um elemento têm de ser exatamente iguais

- **Atributos**

```
<altura unidade="cm">160</altura>
```

onde 'unidade' é o nome do atributo e 'cm' é o valor do atributo. Um atributo especifica uma propriedade de um elemento

- Um elemento pode ter zero, um ou mais atributos
- Os nomes dos atributos são *case-sensitive*
- Um atributo não pode ter dois valores num elemento, ou seja, não pode ser declarado duas ou mais vezes num mesmo elemento
- Os nomes dos atributos não podem possuir aspas, mas os valores tem de ser encapsulados por aspas

- **Referências XML**

```
&amp; ou &#65
```

Há dois tipos de referências, Referências Entidade ou Referências Carácter. A primeira existe por forma a serem representadas por estas referências os caracteres não permitidos no texto anotado visto representarem parte da sintaxe do XML.

Carácter não permitido	Entidade
<	<
>	>
&	&
'	'
"	"

O segundo tipo de referências tem o mesmo intuito mas permite que seja representado qualquer carácter representável em *Unicode*. O número que se segue ao *hashtag* é referente ao código decimal em *Unicode*. Assim `A` representa o carácter 'A'.

- **Texto anotado**

Espaços em branco, *tabs*, ou novas linhas, são ignoradas quando presentes entre elementos e entre atributos. Como já referido há caracteres reservados pelo que se pretende usá-los no texto anotado deve trocar esses caracteres pelas suas referências.

Bibliotecas de conversão

Nesta secção serão abordadas algumas bibliotecas de conversão de [JSON](#) para [XML](#), aquelas com maior popularidade e adesão no [NPM](#).

Por forma a ter uma ideia do resultado devolvido por cada biblioteca será usado o seguinte exemplo:

```
[
  {
    "nome": "Maria",
    "idade": "22",
    "morada": {
      "pais": "Portugal",
      "cidade": "Braga"
    }
  },
  {
    "nome": "João",
    "idade": "24",
    "morada": {
      "pais": "Portugal",
      "cidade": "Vila Real"
    }
  }
]
```

Exemplo 6.2: Exemplo em [JSON](#) a converter

xml - js

Esta biblioteca permite a conversão nos dois sentidos, ou seja, de [JSON](#) para [XML](#) e de [XML](#) para [JSON](#). As principais características que esta biblioteca possui para uma conversão de [JSON](#) para [XML](#) são:

- Mantém a ordem dos elementos
- Totalmente compatível com [XML](#)
- Reversível, é possível reverter o resultado para o original
- Podem ser fornecidas funções personalizadas para processamento adicional para diferentes partes do [XML](#) ou do [JSON](#) a converter

```
<0>
  <nome>Maria</nome>
  <idade>22</idade>
  <morada>
    <pais>Portugal</pais>
    <cidade>Braga</cidade>
  </morada>
</0>
<1>
  <nome>João</nome>
  <idade>24</idade>
  <morada>
    <pais>Portugal</pais>
    <cidade>Vila Real</cidade>
  </morada>
</1>
```

Exemplo 6.3: Resultado da conversão do exemplo 6.2 usando o conversor `xml - js`

xmlbuilder

A biblioteca tem como intuito principal a construção de documentos [XML](#) através do uso de funções, como se pode comprovar de seguida:

```
var builder = require('xmlbuilder');

var xml = builder.create('pessoa')
  .ele('nome', 'Maria')
  .up()
  .ele('idade', {'tipo': 'numero'}, '23')
  .up()
  .ele('morada')
```



```

    .ele('pais', 'Portugal')
    .up()
    .ele('cidade', 'Braga')
    .up()
    .up()
    .ele('altura', {'unidade': 'cm'}, '160')
    .up()
    .com('Isto é um comentário')
    .end({ pretty: true});

```

Exemplo 6.4: Código para a construção em XML do exemplo 6.1 usando o `xmlbuilder`

Além disso, permite a geração do XML a partir de um objeto JSON o que permite, como tal, a conversão de JSON para XML que se necessita.

```

<?xml version="1.0" encoding="utf-8"?>
<nome>Maria</nome>
<idade>22</idade>
<morada>
  <pais>Portugal</pais>
  <cidade>Braga</cidade>
</morada>
<nome>João</nome>
<idade>24</idade>
<morada>
  <pais>Portugal</pais>
  <cidade>Vila Real</cidade>
</morada>

```

Exemplo 6.5: Resultado da conversão do exemplo 6.2 usando o conversor `xmlbuilder`

Na construção do documento a partir de um objecto, quando se pretende que um elemento tenha um atributo, é necessário que o objeto para essa chave possua um objeto em vez de apenas o valor. Nesse objeto, as propriedades começadas por '@' indicam atributos ('@unidade': 'cm' onde 'unidade' é o nome do atributo e 'cm' é o valor do atributo) e a propriedade '#text' representa o valor do elemento.

Há também a possibilidade de sobrescrever as funções usadas para escrever o documento XML (funções de escrita dos elementos, comentários, etc) bem como as funções que convertem os valores (texto anotado, nome de elementos e atributos, etc).

Por fim, convém referir que esta biblioteca já possui uma sucessora, `xmlbuilder2`, que foi redesenhada por forma a estar em total conformidade com a especificação moderna do DOM.

xml2js

De igual forma como a biblioteca `xml-js`, esta biblioteca consegue converter de JSON para XML como de XML para JSON.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<root>
  <nome>Maria</nome>
  <idade>22</idade>
  <morada>
    <pais>Portugal</pais>
    <cidade>Braga</cidade>
  </morada>
  <nome>João</nome>
  <idade>24</idade>
  <morada>
    <pais>Portugal</pais>
    <cidade>Vila Real</cidade>
  </morada>
</root>
```

Exemplo 6.6: Resultado da conversão do exemplo 6.2 usando o conversor `xml2js`

Se se pretende que um determinado elemento possua atributos é necessário que o objeto para esse elemento seja um objeto (`{$: {unidade: 'cm'}, _: '160'}`) onde `'_'` é o texto anotado e os vários atributos devem estar no objeto da propriedade `'$'` em que o nome da propriedade é o nome do atributo e o valor da propriedade é o valor do atributo.

6.1.2 CSV

O **Comma Separated Values (CSV)** como a sua sigla indica é um formato em que os campos de um registo são separados por uma vírgula. Contudo, os campos não precisam de ser obrigatoriamente separados por vírgula já que podem também ser separados por ponto e vírgula, *tabs*, *pipes* (`'|'`) ou ainda outro qualquer carácter que se pretenda usar. Quando se usa o carácter separador ou uma nova linha (`'\n'`) no campo de um registo, por forma a não ser considerado como um separador de campos ou de registos respetivamente, pode-se encapsular os valores por aspas (`""`) sendo que também se pode usar outro carácter em vez das aspas. Já quando se usa o carácter de encapsulamento no campo de um registo este deve ser protegido (escaped) repetindo o mesmo carácter de encapsulamento.

Um documento **CSV** é nada mais que uma tabela que guarda dados em texto. É usado essencialmente para a troca de dados.

Após esta pequena introdução ao **CSV** apresenta-se as regras de sintaxe para a criação de um documento **CSV**:

- Campos separados com um separador, normalmente uma vírgula
- Cada registo deve estar numa linha. Cada registo deve começar numa linha, mas cada registo pode ter várias linhas, desde que os campos multi linha sejam encapsulados por aspas

- Após o último registo não deve estar presente um *carriage return*
- Opcionalmente, na primeira linha do documento, deve estar presente o cabeçalho com os nomes das colunas separados pelo separador usado no resto do documento
- O carácter de encapsulamento deve ser usado para encapsular um campo quando necessário (pode ser usado sempre), isto é, quando o carácter separador ou uma nova linha ('\n') são usados num campo de um registo

```
Nome,Idade,País,Cidade,Altura
"Silva, Maria",23,Portugal,Braga,160cm
```

Exemplo 6.7: Pequeno exemplo em CSV

Bibliotecas de conversão

Nesta secção irá também ser aprofundada algumas bibliotecas de conversão de JSON para CSV, aquelas com maior popularidade no NPM.

`papaparse`

O `papaparse` tem como principal objetivo realizar o *parse* de ficheiros CSV. Consegue contudo reverter, convertendo de JSON para CSV.

```
nome,idade,morada
Maria,22,[object Object]
João,24,[object Object]
```

Exemplo 6.8: Resultado da conversão do exemplo 6.2 usando o conversor `papaparse`

Olhando para o resultado 6.8 consegue-se perceber que o conversor não consegue converter objetos aninhados. Para além disso, apesar de haver a possibilidade de alterar o carácter separador, o carácter de *escape*, o carácter de nova linha bem como definir o nome das colunas no cabeçalho, não é possível definir uma função por forma a alterar a conversão dos campos de cada propriedade.

`json2csv`

Biblioteca totalmente dedicada à conversão de JSON para CSV. Tem como principais características:

- Permite a alteração dos caracteres de separação, de encapsulamento e de nova linha

- *Escape* automático
- Permite escolher que campos devolver (mesmo em objetos aninhados ou listas aninhadas) e que cabeçalho associar a cada campo
- Aceita funções de transformação para cada campo

```
"nome", "idade", "morada.pais", "morada.cidade"
"Maria", "22", "Portugal", "Braga"
"João", "24", "Portugal", "Vila Real"
```

Exemplo 6.9: Resultado da conversão do exemplo 6.2 usando o conversor `json2csv`

Para além disso permite o *unwind* de uma propriedade que possua uma lista, fazendo com que sejam gerados *x* registos de acordo com o tamanho da lista, sendo alterado apenas os valores que provêm da lista. Infelizmente isto acrescenta mais colunas, não permitindo que no caso de os elementos da lista (filhos) tenham a mesma estrutura que o pai, sejam adicionados como novos registos, sem adicionar novas colunas e sem repetir a informação do pai.

Por exemplo, se pretendermos converter o seguinte documento [JSON](#):

```
[
  {
    "nome": "Maria",
    "idade": "22",
    "morada": {
      "pais": "Portugal",
      "cidade": "Braga"
    },
    "filhos": [
      {
        "nome": "António",
        "idade": "2",
        "morada": {
          "pais": "Portugal",
          "cidade": "Braga"
        }
      }
    ]
  },
  {
    "nome": "João",
    "idade": "24",
    "morada": {
      "pais": "Portugal",
      "cidade": "Vila Real"
    },
    "filhos": []
  }
]
```

]

Exemplo 6.10: Outro exemplo em [JSON](#) a converter

Usando este conversor é possível obter:

```
"nome","idade","morada.pais","morada.cidade","filhos.nome","filhos.idade","filhos.morada.pais","filhos.morada.cidade"
"Maria","22","Portugal","Braga","António","2","Portugal","Braga"
"João","24","Portugal","Vila Real",,,,
```

Exemplo 6.11: Resultado da conversão do exemplo [6.10](#) usando o conversor `json2csv`

Quando o resultado pretendido seria por exemplo:

```
"nome","idade","morada.pais","morada.cidade"
"Maria","22","Portugal","Braga",
"António","2","Portugal","Braga"
"João","24","Portugal","Vila Real"
```

Exemplo 6.12: Resultado pretendido da conversão do exemplo [6.10](#)

6.1.3 Ontologia

O que é uma ontologia?

Uma ontologia é uma descrição formal e explícita de conceitos num domínio de discurso (classes, chamado às vezes de conceitos), propriedades de cada conceito que descrevem várias características e atributos do conceito (*slots*, às vezes chamados de funções ou propriedades) e restrições sobre *slots* (facetas, às vezes chamadas de restrições de função). [40] Uma ontologia, juntamente com um conjunto de instâncias individuais de classes, constitui uma base de conhecimento. [40]

Uma ontologia pode ser dividida em duas partes:

- **Estrutura:** Onde é definido as classes, as propriedades (*slots*) e as restrições das propriedades (facetas)
- **População:** Onde é definido indivíduos (de classes) e propriedades dos indivíduos. Esta parte é construída tendo por base a estrutura definida.

De certa forma, a estrutura é como se fosse o modelo definido e a população a informação guardada respeitando o modelo definido.

Falemos agora um pouco mais sobre cada componente de uma ontologia.

Começando pelas classes, estas correspondem a conceitos, sendo que uma classe pode ter subclasses (onde uma subclasse herda as propriedades da sua superclasse) e superclasses. Para além disso, as classes possuem instâncias (ou indivíduos). Quanto às propriedades das classes existem dois tipos:

- Atributos: propriedade que permite definir uma característica de uma classe (ou indivíduo) (p.e: Se pessoa for uma classe, o nome será um atributo)
- Relações: propriedade que permite estabelecer relações entre classes (ou indivíduos) (p.e: A relação de pai entre duas pessoas)

As próprias propriedades podem ser caracterizadas através das restrições. As principais restrições são:

- Cardinalidade: Indica quantos valores pode ter uma propriedade, θ a n (p.e: Uma pessoa só tem um pai biológico.; outro exemplo: Uma pessoa pode ter várias nacionalidades)
- Tipo de valor: Indica que tipo de valor terá a propriedade (atributo), *string*, número, *boolean*, etc (p.e: A idade de uma pessoa é um número; outro exemplo: O nome de uma pessoa é uma *string*)
- *Domain* e *Range*: Indica que tipo de instâncias podem ser relacionadas por uma propriedade. O *Domain* indica a quem pertence a propriedade (atributo ou relação). Já o *Range* indica para uma relação o tipo de instâncias que podem ser relacionadas com o *Domain* (p.e: Na relação pai, o seu *Domain* e o seu *Range* são iguais a Pessoa.; outro exemplo: Seja Animal outra classe, a relação tem dono tem como *Domain* Animal e como *Range* Pessoa; ainda outro exemplo: O *Domain* do atributo nome é Pessoa)

Web Semântica

A *web* semântica é uma extensão da [WWW](#) com o objetivo de tornar a informação da internet legível por máquinas (*machine-readable*).

Para isso a [World Wide Web Consortium \(W3C\)](#) está a ajudar a criar a *stack* tecnológica necessária. O termo *web* semântica é a visão da [W3C](#) acerca da *Linked Data* na *Web*, ou seja, é a *web* dos dados em que a coleção de tecnologias da *web* semântica ([RDF](#), [SPARQL](#), [OWL](#), [SKOS](#), etc) oferece um ambiente onde as pessoas podem criar bases de dados na *web*, construir ontologias e escrever regras para manipular os dados; e onde as aplicações podem questionar esses dados e desenhar inferências usando vocabulários.

Para tornar a *web* dos dados uma realidade é necessário existir uma grande quantidade de dados na *web* num formato *standard* ([RDF](#)), acessível e gerível pelas ferramentas da *web* semântica. Além disso, a *web* semântica necessita também que existam relações entre os dados para criar a *web* dos dados (*Linked Data*). É também importante ser possível configurar *query endpoints* para aceder aos dados mais convenientemente. A [W3C](#) fornece uma paleta de tecnologias, das quais se destaca o [SPARQL](#), para obter acesso aos dados.

A *Linked Data* é usada para a integração em grande escala e o *reasoning*¹ dos dados na *web*.

Na *web* semântica, os vocabulários definem os conceitos e os relacionamentos usados para descrever e representar uma área de interesse. Não há uma divisão clara entre o que é referido como “vocabulários” e “ontologias”. A tendência é usar a palavra “ontologia” para coleções de termos mais complexos e mais formais, enquanto que “vocabulário” nas coleções menos formais. A W3C oferece várias técnicas para descrever e definir diferentes formas de vocabulários num formato *standard*. Estas incluem RDF e RDF Schemas, SKOS, OWL e RIF. A escolha da tecnologia depende da complexidade e do rigor necessário.

Tal como as bases de dados relacionais ou o XML necessitam de linguagens de pesquisa (*query*) específicas (SQL e XQuery, respetivamente), a *web* dos dados, tipicamente representada usando RDF como formato de dados, necessita também de uma linguagem de pesquisa. Isto é providenciado pelo SPARQL, que é também um protocolo de comunicação, permitindo enviar *queries* e receber resultados, por exemplo, através de HTTP ou SOAP.

As *queries* SPARQL são baseadas em padrões (triplos). O RDF pode ser visto como um conjunto de relacionamentos entre recursos (triplos RDF). As *queries* SPARQL possuem um ou mais padrões semelhantes aos triplos RDF, exceto que um ou mais dos recursos constituintes referenciados são variáveis. Um motor SPARQL retornará os recursos que para todos os triplos correspondem a esses padrões. A informação devolvida pelos *endpoints* SPARQL pode ser uma tabela ou por exemplo JSON.

GraphDB

O GraphDB é uma Base de Dados (BD) Semântica baseada em grafos compatível com os padrões W3C. É a base de dados principal quando se pretende trabalhar com a *web* semântica. Suporta RDF e SPARQL e possui funcionalidades de exportação dos triplos presentes numa BD armazenada no GraphDB.

Para um fácil uso e compatibilidade com os *standards* da indústria, o GraphDB implementou as interfaces da *framework* RDF4J, a especificação do protocolo W3C SPARQL² e suporta vários formatos de serialização RDF³. [42]

O GraphDB é um *plugin* SAIL para a *framework* RDF4J fazendo uso extensivo dos recursos e infraestrutura do RDF4J especialmente do modelo RDF, dos *parsers* RDF e dos motores de pesquisa. [41]

Assim, o GraphDB possui uma REST API do servidor RDF4J⁴ a partir da qual é possível obter todos os triplos de uma BD através da rota

<url do GraphDB>/repositories/<id do repositório (BD)>/statements

indicando no cabeçalho HTTP *Accept* o formato de serialização RDF³ de saída (*MIME type*⁵) dos triplos.

¹Em *web* semântica é a inferência de informação (triplos) a partir de um conjunto de factos (triplos), ou seja, a descoberta de novos factos (triplos)

²Ver <https://www.w3.org/TR/sparql11-protocol/>

³TriG, BinaryRDF, TriX, N-Triples, N-Quads, N3, RDF/XML, RDF/JSON, JSON-LD e Turtle

⁴Ver <https://rdf4j.org/documentation/rest-api/>

⁵*Standard* que indica a natureza e o formato de um documento, ficheiro ou conjunto de *bytes*. Ver RFC 6838

6.2 SOLUÇÃO

Nesta secção será apresentado a especificação dos documentos finais das exportações a realizar, decidindo se algumas das bibliotecas apontadas no estado da arte (6.1) permitem auxiliar ou realizar as conversões necessárias.

6.2.1 XML

De seguida, apresenta-se a especificação do documento final em XML:

- Os dados exportados devem ser encapsulados com a *tag root* por forma a garantir que só existe um elemento *root* no documento XML gerado respeitando as regras do XML
- Cada tipo de dados do JSON deve ser convertido da seguinte forma:
 - *string*: Mantém-se igual tirando os caracteres “<”, “>”, “&”, “'” e “”” que devem ser convertidos para a *Entity Reference*⁶ correspondente;
 - *number*: Mantém-se igual;
 - *boolean*: Mantém-se igual;
 - *null*: Origina uma *string* vazia;
 - *array*: Cada item do *array* deve ser encapsulado numa *tag item* que possui um atributo *index* que indica a posição do elemento no *array* e um atributo *type* que indica o tipo do elemento do *array*. O tipo pode ser *number*, *boolean*, *string*, *array* ou *object*;
 - *object*: Para cada propriedade deve ser criado uma *tag* com valor igual à chave da propriedade e ao valor da propriedade deve ser aplicado recursivamente uma das transformações desta lista. Esta *tag* deve ter um atributo *type* em que o seu valor, tal como nos *arrays*, pode ser *number*, *boolean*, *string*, *array* ou *object*.

Depois de compreendida a especificação, se observarmos as bibliotecas exploradas na secção do estado da arte, percebemos que não há nenhuma que permita obter esta especificação sem alterar o objeto JSON a exportar (a converter) o que acaba por em certas situações ser mais complicado do que construir um conversor específico para esta especificação. Assim, decidiu-se que seria criado um conversor de JSON para XML.

⁶“<” para “<”, “>” para “>”, “&” para “&”, “'” para “'” e “”” para “"”

6.2.2 CSV

O documento [CSV](#) exportado deve respeitar a seguinte especificação:

- O conjunto de objetos permitidos é lista de classes, de entidades, de tipologias e de legislações e objeto de classe, de entidade, de tipologia e de legislação;
- A conversão das listas de classes, de entidades, de tipologias e de legislações deve ter a presença dos títulos na primeira linha e depois um elemento por linha;
- Todos os valores das propriedades tem de ser encapsulados com aspas ("");
- Os valores de uma linha devem ser concatenados com ponto e vírgula (;);
- As linhas devem ser concatenadas com nova linha (\n);
- Caso o valor de uma propriedade a converter seja uma lista, a conversão a realizar irá depender da propriedade e do objeto que está a ser convertido:
 - Num objeto Classe:
 - * Propriedade 'notasAp':
 - Título: Notas de aplicação
 - Valor: Concatenação por #\n da propriedade 'nota' de cada elemento da lista
 - * Propriedade 'exemplosNotasAp':
 - Título: Exemplos de NA
 - Valor: Concatenação por #\n da propriedade 'exemplo' de cada elemento da lista
 - * Propriedade 'notasEx':
 - Título: Notas de exclusão
 - Valor: Concatenação por #\n da propriedade 'nota' de cada elemento da lista
 - * Propriedade 'termosInd':
 - Título: Termos Indice
 - Valor: Concatenação por #\n da propriedade 'termo' de cada elemento da lista
 - * Propriedade 'donos':
 - Título: Donos do processo
 - Valor: Concatenação por #\n da propriedade 'sigla' de cada elemento da lista
 - * Propriedade 'participantes', gera duas colunas no [CSV](#):

- Título: Participante no processo
Valor: Concatenação por #\n da propriedade 'sigla' de cada elemento da lista
- Título: Tipo de intervenção do participante
Valor: Concatenação por #\n da propriedade 'participLabel' de cada elemento da lista
- * Propriedade 'processosRelacionados', gera três colunas no CSV:
 - Título: Código do processo relacionado
Valor: Concatenação por #\n da propriedade 'codigo' de cada elemento da lista
 - Título: Título do processo relacionado
Valor: Concatenação por #\n da propriedade 'titulo' de cada elemento da lista
 - Título: Tipo de relação entre processos
Valor: Concatenação por #\n da propriedade 'idRel' de cada elemento da lista
- * Propriedade 'legislacao', gera duas colunas no CSV:
 - Título: Diplomas jurídico-administrativos REF Ids
Valor: Concatenação por #\n da propriedade 'idLeg' de cada elemento da lista
 - Título: Diplomas jurídico-administrativos REF Títulos
Valor: Cada elemento da lista é mapeado para a concatenação da propriedade 'tipo' com a propriedade 'numero' com um espaço entre as duas propriedades; Concatenação por #\n do mapeamento de cada elemento da lista
- * Propriedade 'filhos': cada elemento deve ser convertido como se tratasse de um objeto classe; Devem ser ignorados os títulos gerados, mantendo apenas os valores numa nova linha do CSV.
- Num objeto Entidade:
 - * Propriedade 'dono':
 - Título: Dono no processo
Valor: Concatenação por #\n da propriedade 'codigo' de cada elemento da lista
 - * Propriedade 'participante', gera duas colunas no CSV:
 - Título: Participante no processo
Valor: Concatenação por #\n da propriedade 'codigo' de cada elemento da lista
 - Título: Tipo de intervenção no processo
Valor: Concatenação por #\n da propriedade 'tipoPar' de cada elemento da lista
 - * Propriedade 'tipologias':

- Título: Tipologias da entidade
Valor: Concatenação por #\n da propriedade 'sigla' de cada elemento da lista
- Num objeto Tipologia:
 - * Propriedade 'entidades':
 - Título: Entidades da tipologia
Valor: Concatenação por #\n da propriedade 'sigla' de cada elemento da lista
 - * Propriedade 'dono':
 - Título: Dono no processo
Valor: Concatenação por #\n da propriedade 'codigo' de cada elemento da lista
 - * Propriedade 'participante', gera duas colunas no CSV:
 - Título: Participante no processo
Valor: Concatenação por #\n da propriedade 'codigo' de cada elemento da lista
 - Título: Tipo de intervenção no processo
Valor: Concatenação por #\n da propriedade 'tipoPar' de cada elemento da lista
- Num objeto Legislação:
 - * Propriedade 'entidades':
 - Título: Entidades
Valor: Concatenação por #\n da propriedade 'sigla' de cada elemento da lista
 - * Propriedade 'regula':
 - Título: Regula processo
Valor: Concatenação por #\n da propriedade 'codigo' de cada elemento da lista
- Na propriedade 'pca' de um objeto Classe:
 - * Propriedade 'justificacao', gera duas colunas no CSV:
 - Título: Critério PCA
Valor: Concatenação por #\n da propriedade 'tipold' de cada elemento da lista
 - Título: ProcRefs/LegRefs PCA
Valor: Cada elemento da lista é mapeado para a concatenação por #\n da lista presente na propriedade 'processos' ou na propriedade 'legs' sendo a concatenação encapsulada por parênteses curvos; Concatenação por #\n do mapeamento de cada elemento da lista
- Na propriedade 'df' de um objeto Classe:

- * Propriedade 'justificacao', gera duas colunas no CSV:
 - Título: Critério DF
Valor: Concatenação por #\n da propriedade 'tipold' de cada elemento da lista
 - Título: ProcRefs/LegRefs DF
Valor: Cada elemento da lista é mapeado para a concatenação por #\n da lista presente na propriedade 'processos' ou na propriedade 'legs' sendo a concatenação encapsulada por parênteses curvos; Concatenação por #\n do mapeamento de cada elemento da lista
- Caso o valor de uma propriedade seja um objeto, as propriedades do objeto aninhado devem ser processadas como se tratassem de propriedades do objeto possuidor da propriedade com o objeto aninhado
- Nos casos em que o valor não é uma lista nem um objeto deve ser mantido o valor (apenas encapsulado por ") e associado o seguinte título:
 - Num objeto Classe:
 - * Propriedade 'codigo': Código
 - * Propriedade 'titulo': Título
 - * Propriedade 'descricao': Descrição
 - * Propriedade 'tipoProc': Tipo de processo
 - * Propriedade 'procTrans': Processo transversal (S/N)
 - Num objeto Entidade:
 - * Propriedade 'sigla': Sigla
 - * Propriedade 'designacao': Designação
 - * Propriedade 'estado': Estado
 - * Propriedade 'sioe': ID SIOE
 - * Propriedade 'internacional': Internacional
 - Num objeto Tipologia:
 - * Propriedade 'sigla': Sigla
 - * Propriedade 'designacao': Designação
 - * Propriedade 'estado': Estado
 - Num objeto Legislação:

- * Propriedade 'tipo': Tipo
- * Propriedade 'numero': Número
- * Propriedade 'data': Data
- * Propriedade 'sumario': Sumário
- * Propriedade 'fonte': Fonte
- * Propriedade 'link': Link
- Na propriedade 'pca' de um objeto Classe:
 - * Propriedade 'valores': Prazo de conservação administrativa
 - * Propriedade 'notas': Nota ao PCA
 - * Propriedade 'formaContagem': Forma de contagem do PCA
 - * Propriedade 'subFormaContagem': Sub Forma de contagem do PCA
- Na propriedade 'df' de um objeto Classe:
 - * Propriedade 'valor': Destino Final
 - * Propriedade 'notas': Notas ao DF
- Na exportação para Excel as concatenações #\n devem ser apenas #
- As propriedades não referidas nesta especificação devem ser ignoradas

A partir da especificação e olhando para as bibliotecas de conversão exploradas no estado da arte chegou-se à conclusão que não havia nenhuma biblioteca que contivesse todas as funcionalidades necessárias. Contudo, é importante assinalar que a biblioteca `json2csv` possui grande parte das funcionalidades necessárias permitindo respeitar grande parte da especificação. Apesar disso, não é possível respeitar *“Propriedade ‘filhos’: cada elemento deve ser convertido como se tratasse de um objeto classe; Devem ser ignorados os títulos gerados, mantendo apenas os valores numa nova linha do CSV”*.

Além disso, o uso da biblioteca em questão obrigaria à criação de várias funções de transformação de dados por forma a respeitar a especificação. Por estas duas razões, decidiu-se que era mais conveniente, mais rápido e mais fácil criar um conversor próprio de `JSON` para `CSV`.

6.2.3 Ontologia

Por fim quanto à exportação da ontologia, das três é a mais simples visto que o `GraphDB` como já se referiu no estado da arte, possui funcionalidades de exportação dos triplos de uma `BD`. Assim, apenas é

necessário realizar um pedido ao *GraphDB* da [API](#) de dados indicando no cabeçalho *Accept* do pedido o formato de saída.

Esta rota da [API](#) do *GraphDB* possui vários formatos de exportação pelo que se decidiu suportar na [API](#) da [CLAV](#) apenas os mais populares.

Portanto, dos vários formatos de serialização [RDF](#) serão apenas suportados (acessíveis) na [CLAV](#), o [Turtle](#) (`text/turtle`), o [JSON-LD](#) (`application/ld+json`) e o [RDF/XML](#) (`application/rdf+xml`).

Apesar da facilidade de exportação da ontologia estes pedidos de exportação originam um grande consumo de recursos de *hardware* por parte do *GraphDB* visto que cada pedido devolve todos os triplos de uma [BD](#) (a atual [BD](#) da [CLAV](#) possui já cerca de 150 000 triplos explícitos e cerca de 85 000 triplos implícitos) para além da conversão necessária desses triplos para o formato de serialização [RDF](#) de saída. Deve-se então limitar o número de pedidos de exportação realizados ao *GraphDB*. Para tal irá ser usado o seguinte mecanismo de controlo/*cache*:

- Os ficheiros exportados são mantidos pela [API](#) da [CLAV](#);
- Mantém-se dois ficheiros por cada serialização [RDF](#), um com os triplos explícitos e outro com os triplos explícitos e implícitos;
- Se o ficheiro pretendido não existe na [API](#) da [CLAV](#) realiza-se o pedido de exportação ao *GraphDB*;
- Se o ficheiro pretendido existe na [API](#) da [CLAV](#) mas não é atualizado há sete dias realiza-se o pedido de exportação ao *GraphDB*;
- Se o ficheiro pretendido existe na [API](#) da [CLAV](#) e foi atualizado há menos de sete dias devolve-se ao utilizador o ficheiro guardado na [API](#) da [CLAV](#);
- Mantém-se na [API](#) da [CLAV](#) apenas o ficheiro mais recente para cada versão de cada serialização [RDF](#);
- Cada ficheiro é apenas atualizado (removendo o antigo) quando é feito um pedido por um utilizador desse ficheiro.

Assim, respeitando todas estas restrições, são mantidas pela [API](#) da [CLAV](#) no máximo seis ficheiros, dois por cada serialização [RDF](#). Para além disso estes ficheiros são atualizados no melhor caso de sete em sete dias e no pior caso nunca se o ficheiro nunca for requisitado pelos utilizadores.

6.2.4 Exportação na [API](#) de dados

Nesta secção será explicado de que forma será possível exportar os dados da [API](#). Para tal definiu-se a *query string* `f s` (formato de saída) onde é possível indicar claro está o formato de saída. Esta *query string*

estará presente nas rotas onde será possível exportar os dados. Para além disso, nestas rotas também se pode indicar o formato de saída através do cabeçalho `Accept`.

De seguida são apresentadas as rotas onde é possível realizar exportação, os formatos de saída disponíveis para cada rota bem como os valores a usar de forma a obter uma exportação nesse formato:

Rota	Formato de saída (valor a usar)
GET /<versãoAPI>/classes	<ul style="list-style-type: none"> • JSON (<code>application/json</code>) • XML (<code>application/xml</code>) • CSV (<code>text/csv</code> ou ainda <code>excel/csv</code> se se pretender o CSV no formato para o <i>Excel</i>)
GET /<versãoAPI>/classes/:id	
GET /<versãoAPI>/entidades	
GET /<versãoAPI>/entidades/:id	
GET /<versãoAPI>/tipologias	
GET /<versãoAPI>/tipologias/:id	
GET /<versãoAPI>/legislacao	
GET /<versãoAPI>/legislacao/:id	
GET /<versãoAPI>/ontologia	<ul style="list-style-type: none"> • Turtle (<code>text/turtle</code>) • JSON-LD (<code>application/ld+json</code>) • RDF/XML (<code>application/rdf+xml</code>)

Tabela 3: Rotas com exportação, formatos de saída disponíveis para cada rota e valores a usar por forma a exportar nesse formato de saída

Portanto, por exemplo para obter as Classes em **CSV** basta realizar o seguinte pedido à **API**:

```
GET /<versãoAPI>/classes?fs=text/csv
```

Já em termos de fluxo dos dados durante esta exportação, para as 8 primeiras rotas da tabela 3 inicialmente os dados são obtidos da **BD** (*GraphDB*) ou da *cache* da **API**. Caso o formato de saída seja **JSON** é devolvido ao utilizador sem qualquer conversão. Caso contrário os dados são convertidos através de um dos conversores já descritos para o formato de saída apropriado. Na última rota, a da exportação da ontologia, a informação é devolvida no formato apropriado pela própria **BD** (*GraphDB*) ou da *cache* referida em 6.2.3 de acordo com o pedido.

6.3 IMPLEMENTAÇÃO

Um dos requisitos da **API** da **CLAV** é permitir a exportação de Classes, Entidades, Tipologias e Legislações em formato **JSON**, **XML** e **CSV**. Deve também permitir exportar toda a ontologia do projeto nos formatos **Turtle**, **JSON-LD** e **RDF/XML**.

Para a primeira parte foi necessário desenvolver dois conversores, de **JSON** para **XML** e de **JSON** para **CSV** visto que o **JSON** já é por predefinição devolvido.

6.3.1 XML

O conversor de **JSON** para **XML** criado funciona para qualquer estrutura em **JSON**.

Se por exemplo tivermos o seguinte **JSON** a converter:

```
{
  "nivel": 2,
  "codigo": "100.10",
  "titulo": "Elaboração de diplomas jurídico-normativos",
  "notasAp": [
    {
      "idNota": "http://jcr.di.uminho.pt/m51-clav#na_c100.10_MRIKl-RBu_2sz5u9FzPqH",
      "nota": "Qualquer despacho com diretrizes gerais e abstratas"
    }
  ],
  "subdivisao4Nivel01Sintetiza02": true,
  "pca": {
    "valores": "",
    "notas": "",
    "justificacao": []
  },
  "df": {
    "valor": "NE",
    "nota": null,
    "justificacao": []
  }
}
```

Exemplo 6.13: **JSON** exemplo a converter

O resultado com este conversor será:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <nivel type="number">2</nivel>
  <codigo type="string">100.10</codigo>
  <titulo type="string">Elaboração de diplomas jurídico-normativos</titulo>
  <notasAp type="array">
    <item index="0" type="object">
      <idNota type="string">http://jcr.di.uminho.pt/m51-clav#na_c100.10_MRIKl-RBu_2sz5u9FzPqH</idNota>
      <nota type="string">Qualquer despacho com diretrizes gerais e abstratas</nota>
    </item>
  </notasAp>
  <subdivisao4Nivel01Sintetiza02 type="boolean">true</subdivisao4Nivel01Sintetiza02>
  <pca type="object">
    <valores type="string"></valores>
    <notas type="string"></notas>
    <justificacao type="array">
      </justificacao>
    </pca>
  <df type="object">
```



```
<valor type="string">NE</valor>
<nota type="object">
</nota>
<justificacao type="array">
</justificacao>
</df>
</root>
```

Exemplo 6.14: XML resultante da conversão do JSON presente em 6.13

6.3.2 CSV

Da mesma forma que o XML, o CSV é convertido sem recurso a uma biblioteca que converta já de si o JSON para CSV visto que cada objeto JSON a exportar necessita de uma exportação personalizada para CSV. Ao contrário do conversor desenvolvido para XML, o conversor para CSV não converte qualquer objeto para CSV mas apenas um conjunto restrito de objetos JSON.

O conjunto de objetos permitidos é lista de classes, de entidades, de tipologias e de legislações, objeto de classe, de entidade, de tipologia e de legislação e mais algumas estruturas especiais de classes.

Quanto à conversão em si, possui uma estrutura interna durante a conversão. Esta estrutura é uma lista de listas, em que cada lista representa uma linha do CSV. Cada elemento de uma das listas representará uma célula do CSV. A primeira lista será a primeira linha do CSV e como tal possuirá os títulos. As restantes listas serão as linhas seguintes do CSV em que cada elemento possuirá os valores já transformados em *strings* dos campos dos objetos.

Para além desta estrutura interna existe um dicionário que permite agilizar o algoritmo de conversão. Este dicionário possuirá vários dicionários, um por cada objeto (Classe, Entidade, Tipologia e Legislação) em que cada um destes dicionários irá ter como chaves os campos a converter. Para cada um destes campos existe um tuplo em que na primeira posição está presente o título a colocar no CSV referente a este campo e na segunda posição a função de transformação a executar para o valor do campo. Há a presença de três casos especiais:

- Quando o valor do campo é uma lista de objetos e pretendemos apenas um dos campos de cada objeto, o valor do campo deve ser `campo^campoDoObjeto` e deve ser usada a função de transformação `map_value(<campoDoObjeto>)`. Pode ser usada outra função de transformação onde recebe uma lista como argumento e deve retornar apenas uma *string*. Isto é útil para casos em que se pretende combinar vários campos numa *string* para cada objeto da lista.
- Quando o valor do campo é um objeto do qual irá resultar vários títulos, na primeira posição do tuplo deve estar presente uma *string* vazia e a função de transformação deve devolver uma lista com duas posições, na primeira com os títulos e na segunda com os valores transformados dos campos

- Quando o valor do campo é uma lista de objetos Classe, Entidade, Tipologia ou Legislação a primeira posição do tuplo deve ser `null` e a função de transformação deve devolver uma lista de listas sem a primeira linha de títulos

No caso da conversão de um objeto e consoante a transformação (ou seja, o título do dicionário) a inserção realizada na lista de listas varia:

- `título == null`: concatena-se a lista de listas devolvida pela função de transformação à lista de listas
- `título == ""`: concatena-se a lista dos elementos da primeira linha devolvida pela função de transformação com os elementos da primeira linha e realiza-se o mesmo para o caso da segunda linha devolvida, concatena-se a segunda linha com a segunda linha
- Nos restantes casos protege-se⁷ o título presente no dicionário e adiciona-se à primeira lista; para além disso, o valor transformado devolvido pela função de transformação é adicionado já protegido à segunda lista.

No caso da conversão de uma lista de objetos, para cada objeto será feita a conversão já apresentada para um objeto, onde depois é ignorada a linha dos títulos em todos os objetos exceto no primeiro objeto da lista onde é mantido os títulos gerados. Ou seja, na primeira linha estará presente os títulos e nas seguintes linhas, em cada linha estará presente os valores de um objeto.

O último passo seja para uma lista ou para um único objeto é transformar a estrutura interna no [CSV](#). Para tal, os elementos de cada lista da lista são juntos de acordo com um separador (neste caso é usado o ponto e vírgula, “;”) tornando a lista de listas numa lista de *strings*. Por fim, as *strings* desta lista são juntas através da inserção de novas linhas (“\n”) entre cada *string* gerando o [CSV](#) final.

De seguida apresenta-se um exemplo de uma conversão, onde o ficheiro [JSON](#) a converter é o mesmo usado para exemplificar a conversão de [XML](#) presente em 6.13.

```
"Código";"Título";"Notas de aplicação";"Prazo de conservação administrativa";"Nota ao PCA";"Forma de contagem do
PCA";"Sub Forma de contagem do PCA";"Critério PCA";"ProcRefs/LegRefs PCA";"Destino final";"Notas ao DF";"
Critério DF";"ProcRefs/LegRefs DF"
"100.10";"Elaboração de diplomas jurídico-normativos";"Qualquer despacho com diretrizes gerais e abstratas
";"";"";"";"";"";"";"";"NE";"";"";""
```

Exemplo 6.15: [CSV](#) resultante da conversão do [JSON](#) presente em 6.13

⁷colocar valor entre aspas (“”)

6.3.3 Exportação na API de dados

As conversões de **JSON** para **XML** ou **CSV** são realizadas por um *middleware* após a obtenção dos dados em **JSON**. Ou seja, primeiro obtêm-se os dados pretendidos e só depois é que é feita a conversão para o formato de saída pretendido permitindo assim uma maior facilidade de desenvolvimento. Além disso, apenas nas rotas onde for usado este *middleware* é que há a possibilidade de exportação nestes formatos, não sendo garantida para o caso do **CSV** já que não suporta todos os objetos **JSON**.

Caso uma determinada conversão não seja suportada pelo conversor **CSV**, esse suporte pode ser adicionado através da edição do dicionário já referido.

Nas rotas, antes de passar para o *middleware* de conversão, o objeto a converter deve ser armazenado em `res.locals.dados` e o tipo deste objeto (por exemplo “entidade”) ser definido em `res.locals.tipo` por forma aos conversores, mais especificamente o conversor de **JSON** para **CSV**, saber que objeto se pretende converter.

6.3.4 Interface de Exportação

Para facilitar a exportação dos dados foi criada uma página de exportação na interface da **CLAV** como se pode observar na figura 16. Esta página permite definir todos os parâmetros tal como através da **API** de dados e, adicionalmente, o *encoding* de saída dos ficheiros. Assim esta interface permite exportar Classes, Entidades, Tipologias, Legislações e a Ontologia.

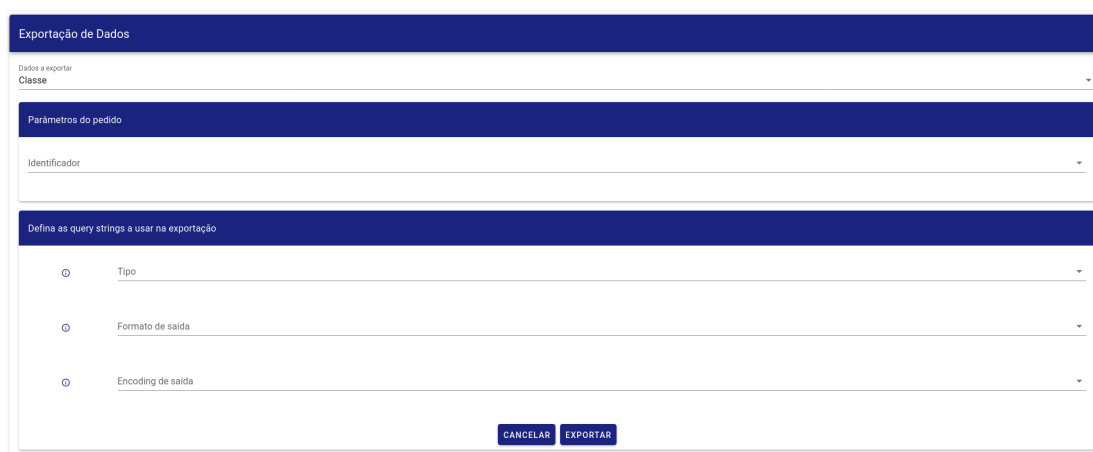


Figura 16: Interface de exportação

Esta página de exportação está acessível em <https://clav.dglab.gov.pt/exportar>.

6.4 RESUMO

Recapitulando, neste capítulo são inicialmente aprofundadas, no estado da arte, algumas bibliotecas disponíveis no [NPM](#) que têm como objetivo exportar de [JSON](#) para [XML](#) ou de [JSON](#) para [CSV](#). É por fim, investigado como poderá ser exportada a ontologia (exportação para [RDF](#)) a partir do *GraphDB*.

Já na secção da solução são apresentados as especificações das conversões a realizar bem como escolhidos que conversores serão usados enquanto que na secção da implementação são apresentados exemplos das conversões que os conversores realizam.

MIGRAÇÃO DE HTTP PARA HTTPS

O [Hypertext Transfer Protocol \(HTTP\)](#) possui várias vulnerabilidades de segurança entre as quais *man-in-the-middle attack*¹ bem como a possibilidade de *eavesdropping*² e *tampering*³ da comunicação entre cliente e servidor.

Com o intuito principal de superar estas vulnerabilidades foi criada a extensão ao [HTTP](#) o [Hypertext Transfer Protocol Secure \(HTTPS\)](#). Este protocolo de comunicação é encriptado através do uso de [Transport Layer Security \(TLS\)](#) ou através do uso do já *deprecated*, por razões de segurança, [Secure Sockets Layer \(SSL\)](#). O [HTTPS](#) oferece autenticação dos *websites* acedidos bem como privacidade e integridade dos dados trocados.

É assim de extrema importância a migração do atual [HTTP](#) para [HTTPS](#) tanto na [API](#) da [CLAV](#) bem como na interface da [CLAV](#).

Na próxima secção é aprofundado o principal [Certificate Authority \(CA\)](#) gratuito para a obtenção de certificados por forma a ativar o [HTTPS](#) na plataforma [CLAV](#).

7.1 ESTADO DA ARTE

7.1.1 *Let's Encrypt*

Para ativar o [HTTPS](#) num website ou numa [API](#), ou seja num domínio, é necessário um [Certificate Authority \(CA\)](#) de onde obter os certificados.

Para o caso da [CLAV](#) necessita-se de um [CA](#) gratuito e onde seja possível gerar certificados [Domain Validation \(DV\)](#) (garantem apenas a validade do domínio e da informação trocada entre utilizador e domínio [8]).

Assim, a escolha recaiu no *Let's Encrypt*, um dos [CA's](#) mais populares. O *Let's Encrypt* foi criado pela [Linux Foundation](#) é gratuito e *open-source*. Além disso, tem como patrocinadores/doadores empresas como a [Mozilla](#), a [Cisco](#), o [Google Chrome](#), o [Facebook](#), entre outros.

¹Ver https://owasp.org/www-community/attacks/Man-in-the-middle_attack

²Ato de ouvir de forma secreta ou furtiva conversas ou comunicações particulares de outras pessoas sem o consentimento destas

³Alteração deliberada ou adulteração dos dados enviados entre cliente e servidor

Para gerar/renovar o certificado **DV** é necessário demonstrar que se possui o controlo sobre o domínio. Com o *Let's Encrypt* isto é efetuado através do uso do protocolo **Automatic Certificate Management Environment (ACME)** que necessita que se corra um agente de gestão de certificados (cliente **ACME**) no servidor do domínio a gerar o certificado. Assim é possível gerar/renovar certificados automaticamente sem a intervenção humana. Esta automatização é importante porque os certificados do *Let's Encrypt* têm apenas a validade de 3 meses pelo que, se fosse necessário fazer manualmente teria de ser feito pelo menos de 3 em 3 meses.

A escolha do cliente **ACME** a usar irá depender se se tem acesso à *shell* da máquina do servidor. Em caso afirmativo o *Let's Encrypt* recomenda o uso do cliente *Certbot* [14].

Caso não se tenha acesso à *shell* irá depender se o provedor de *hosting* suporta o *Let's Encrypt* ou não. Se suportar então é fácil obter o certificado através do provedor. Caso contrário, pode-se pedir ao provedor o suporte (que não resolve de imediato o problema nem é de resolução garantida) ou se o provedor permitir o *upload* de certificados é possível através do *Certbot* gerar um certificado em modo manual mas que acarreta efetuar esta tarefa pelo menos de 3 em 3 meses para renovar o certificado pelo que não é recomendado.

Uma pequena chamada de atenção, o *Let's Encrypt* possui *Rate Limits* [13] pelo que num ambiente de testes deve ser usado o ambiente de testes do *Let's Encrypt*⁴ e não o de produção.

Validação do Domínio

Nesta secção será explicado como é realizada a validação do domínio para a geração/renovação/revogação dos certificados.

Na primeira vez que o agente (cliente **ACME**) interage com o *Let's Encrypt* gera um novo par de chaves e prova ao *Let's Encrypt* que o servidor controla um ou mais domínios. [12]

Para iniciar o processo de validação, o cliente **ACME** questiona o *Let's Encrypt* para que lhe indique o que necessita fazer para provar que controla o domínio [12]. Assuma-se que queremos validar o domínio `example.com`. O *Let's Encrypt* irá gerar um ou mais conjuntos de desafios após olhar para o nome do domínio.

Há duas formas (desafios) do cliente **ACME** provar que controla o domínio perante o *Let's Encrypt* [12]:

- Providenciar um **DNS record** sobre `example.com`
- Providenciar um recurso **HTTP** num **URI** conhecido em `http://example.com/` (é colocado normalmente em `http://example.com/.well-known/acme-challenge`)

Juntamente com os desafios, o *Let's Encrypt* envia um *nonce* (*string* usada uma única vez) que o cliente deve assinar com a chave privada por forma a garantir que este controla o par de chaves evitando *replay attacks*⁵.

⁴Ver <https://letsencrypt.org/docs/staging-environment/>

⁵Para mais informação ver <https://www.kaspersky.com/resource-center/definitions/replay-attack>

Veja-se agora, na figura 17, um exemplo concreto em que o cliente ACME usa o segundo desafio para provar que controla o domínio (`example.com`):

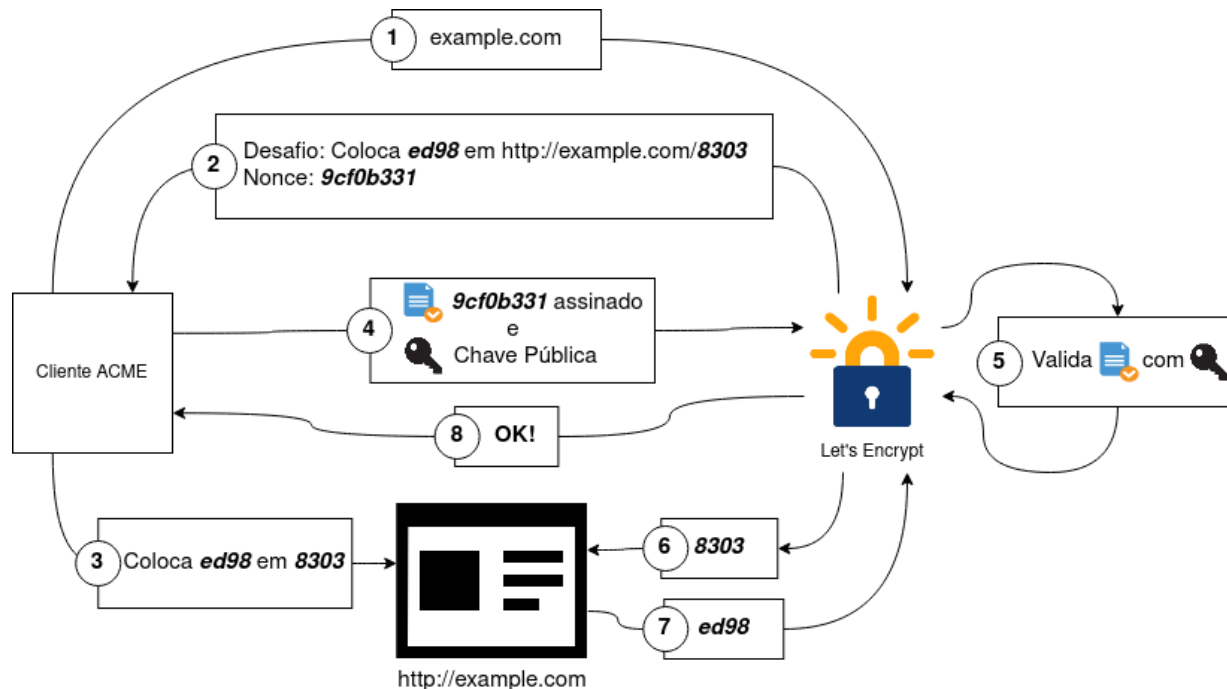


Figura 17: Exemplo de validação do domínio pelo *Let's Encrypt* com sucesso

Como o desafio teve sucesso e o *nonce* assinado pelo cliente ACME é válido então o cliente ACME identificado pela chave pública está autorizado a gerir os certificados para o domínio `example.com`. Assim, para gerar/renovar/revogar o certificado para o domínio basta enviar mensagens de gestão do certificado assinadas com o chave privada (a que assinou o *nonce*) para o *Let's Encrypt*.

Cliente *acme.sh*

Apesar do cliente recomendado pelo *Let's Encrypt* ser o *Certbot* este necessita de ter permissões *root* bem como tem várias dependências para ser executado. Já o cliente *acme.sh*⁶ é uma *script bash* que para a sua instalação basta efetuar *download* da *script* deste. Além disso, não necessita de quaisquer dependências nem de acesso *root* (apenas necessita em casos especiais). A sua utilização, tal como o *Certbot*, é feita através da execução de comandos *bash*.

Para instalar basta:

```
curl https://get.acme.sh | sh
#ou
wget -O - https://get.acme.sh | sh
```

⁶Ver <https://github.com/acmesh-official/acme.sh>

Esta instalação irá ativar desde logo a auto renovação dos certificados através da criação de um *cron job*⁷ pelo que o cliente *acme.sh* irá correr periodicamente para verificar e renovar os certificados se necessário.

Para obter um certificado executa-se:

```
acme.sh --issue -d <dominio> -w <pasta web root>
```

Após obter o certificado, este pode ser instalado no *Nginx* com:

```
acme.sh --install-cert -d example.com \
  --key-file /path/to/keyfile/in/nginx/key.pem \
  --fullchain-file /path/to/fullchain/nginx/fullchain.pem \
  --reloadcmd "nginx -s reload"
```

Neste último comando, o ficheiro de configuração do *Nginx* tem de estar à espera que os ficheiros do certificado estejam em `/path/to/keyfile/in/nginx/key.pem` e em `/path/to/fullchain/nginx/fullchain.pem`.

7.2 SOLUÇÃO

Por forma a realizar a migração de **HTTP** para **HTTPS** há um conjunto de requisitos a cumprir:

- Usar um **CA** gratuito;
- Os certificados devem ser renovados automaticamente;
- Permitir criar uma *script* de automatização para o *deployment*;
- Possuir as seguintes recomendações de segurança:
 - Redirecionar os pedidos **HTTP** para **HTTPS** por forma a impedir que os utilizadores usem uma conexão insegura bem como evitar que alguém se faça passar pela **CLAV** em **HTTP**;
 - Adicionar o cabeçalho **HSTS** recomendado [51, 49];
 - Adicionar vários cabeçalhos e configurar o *reverse proxy* por forma a tornar o **HTTPS** mais forte e a **API**/interface mais segura. Ver [17, 1, 54, 15, 55];
 - Adicionar o cabeçalho **Content Security Policy (CSP)** [18, 20].

Para realizar esta migração a primeira decisão a tomar é o **Certificate Authority (CA)** de onde iremos comprar/obter os certificados. Existem vários **CAs** mas visto termos a restrição de que este deve ser gratuito apenas nos sobra uma alternativa bastante popular, o *Let's Encrypt*. O único revês de usar o *Let's Encrypt* é o facto de os certificados terem uma validade de apenas 90 dias.

Após tomada a decisão de utilizar o **CA Let's Encrypt** é necessário decidir que cliente *Let's Encrypt* usar. Este cliente permite a obtenção e renovação de certificados. Existem vários clientes⁸ dos quais

⁷Mais informação em <https://www.ostechnix.com/a-beginners-guide-to-cron-jobs/>

⁸Ver <https://letsencrypt.org/docs/client-options/>

o *Let's Encrypt* recomenda o *Certbot*⁹. Contudo para usar *Certbot* é necessário ter permissões root (sudo) no servidor bem como é necessário instalar algumas dependências. Por tais razões foi usado o `acme.sh` (Ver 7.1.1). O `acme.sh` é quem irá tratar de toda a gestão dos certificados, renovando-os quando necessário (a renovação é feita a cada 60 dias).

Tendo em conta os requisitos a solução passa por colocar um *reverse proxy* em *Nginx* à frente da *API* de dados, algo que já não é necessário na interface visto este (*Nginx*) já estar presente. Após isso, a solução passará por configurar o *Nginx*, tanto na *API* de dados como na interface, com as várias recomendações de segurança. Passa também por automatizar o *deployment* através de *Docker* e *Docker-Compose* bem como a criação de algumas pequenas *scripts* para:

- gerar certificado local autoassinado com o único objetivo de permitir o primeiro inicio do *Nginx* (ainda não foram gerados os certificados dos domínios) para ser possível o *Let's Encrypt* validar o controlo sobre o domínio (ver 7.1.1) permitindo a geração dos certificados necessários;
- automatizar a instalação do cliente *ACME acme.sh* (*download* e instalação);
- gerar o primeiro certificado para os domínio(s) pretendido(s) com o `acme.sh`;
- instalar o primeiro certificado no caminho apropriado com o `acme.sh` de onde o *reverse proxy* o irá obter;
- gerar *DH parameters* mais fortes para a troca de chaves com recurso ao *OpenSSL*;
- instalar dependências no *container* onde se encontra o *Nginx*:
 - `openssl` para a geração de um certificado local autoassinado e dos *DH parameters*;
 - `cron` para o `acme.sh` criar um *cron job* diário para a verificação/renovação do certificado;
 - `curl` para fazer *download* do `acme.sh`.

A arquitetura a desenvolver com *HTTPS*, presente na figura 18, é semelhante à presente na figura 2 tendo como única diferença a adição do *Nginx* à frente da *API* de dados:

⁹Ver <https://certbot.eff.org/>

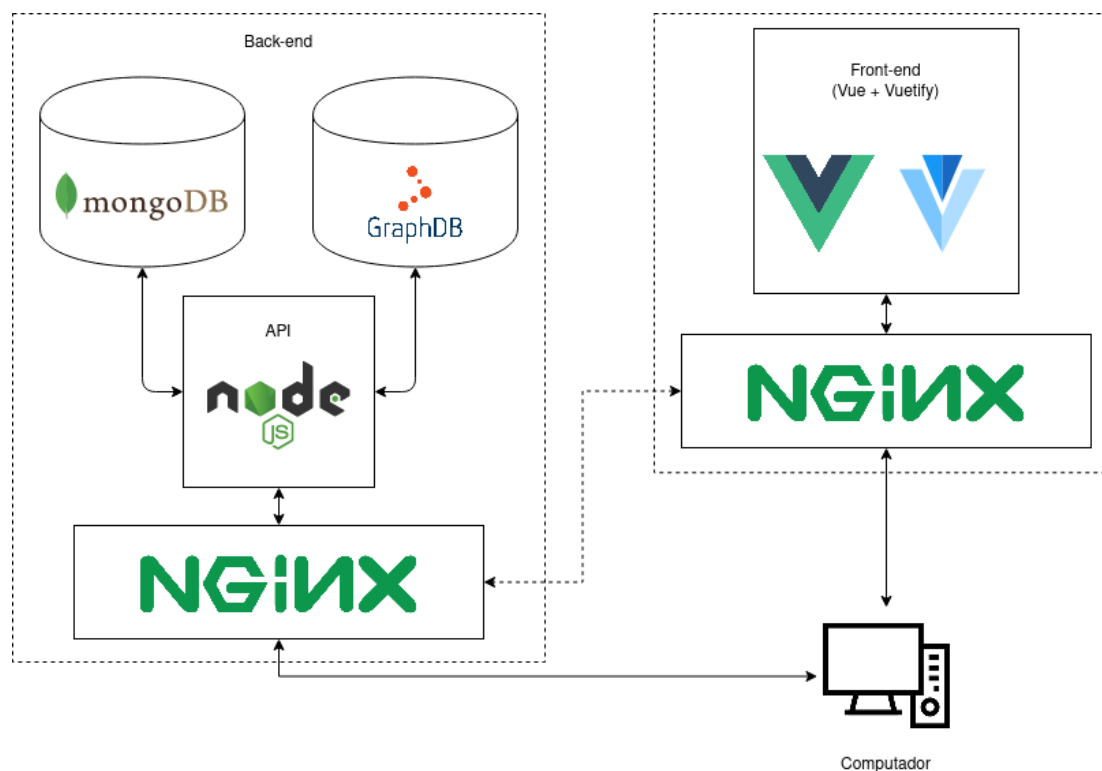


Figura 18: Arquitetura a desenvolver com HTTPS

A ligação a tracejado entre os dois *Nginx*'s pode ou não existir, irá depender se se pretende ou não que o servidor da interface reencaminhe pedidos para o servidor da API de dados.

7.3 IMPLEMENTAÇÃO

A partir dos requisitos já enunciados em 7.2 foram criadas duas *scripts* uma para correr antes de iniciar o *Nginx* e outra para correr depois de o *Nginx* iniciar.

A *script* que corre antes de iniciar o *Nginx* instala o `openssl`, gera o certificado autoassinado e *DH parameters* para permitir o *boot* do *Nginx*.

Já a *script* que corre após o início do *Nginx* realiza o *download* do `acme.sh`, instala-o, obtém o primeiro certificado para o(s) domínio(s) com o `acme.sh`, instala o certificado com o `acme.sh`, gera *DH parameters* mais fortes e, por fim, reinicia o *Nginx* para que o novo certificado e *DH parameters* tenham efeito, ou seja, sejam usados pelo *Nginx*.

Quanto à configuração do *Nginx* entre a da API de dados e a da interface há poucas diferenças. Estas diferenças são que na configuração *Nginx* na API de dados é encaminhado os pedidos para o servidor em *Node.js* enquanto que na configuração *Nginx* da interface há duas variantes onde só uma é usada:

- Apenas serve os ficheiros estáticos da interface

- Serve os ficheiros estáticos da interface e reencaminha os pedidos para a [API](#) de dados, aqueles em que o caminho começa em `/<versão_api>/` ou é igual a `/clav.yaml`.

Quanto às configurações comuns estas serão apresentadas de seguida, explicando para que servem.

Para cumprir o requisito de redirecionamento dos pedidos de [HTTP](#) para [HTTPS](#) é colocado o seguinte bloco de código na configuração:

```
server {
    listen <Porta HTTP>;
    server_name localhost;

    location /.well-known/acme-challenge/ {
        alias /var/www/html/.well-known/acme-challenge/;
    }

    location / {
        return 301 https://$host$request_uri;
    }
}
```

Exemplo 7.1: Redirecionamento de [HTTP](#) para [HTTPS](#) e validação do domínio na configuração *Nginx*

Além disso, permite a validação do controlo do domínio por parte do *Let's Encrypt* visto que este pequeno excerto permite que o *Nginx* receba pedidos em `/.well-known/acme-challenge/`, caminho este onde é colocado a resposta de um desafio do *Let's Encrypt*.

Fica assim definido o que se faz quando se recebe um pedido [HTTP](#). Para os pedidos [HTTPS](#) é criado também um bloco `server` onde é ativado o [HTTP2](#) e é indicado os ficheiros com o certificado:

```
server {
    listen <Porta HTTPS> ssl http2;
    server_name localhost;

    ssl_certificate $CERTS/fullchain.pem;
    ssl_certificate_key $CERTS/key.pem;
    ...
}
```

Exemplo 7.2: Certificado na configuração *Nginx*

Depois são adicionadas várias configurações recomendadas [17, 1, 54, 15, 55, 18, 20, 51, 49]:

```
server {
    ...

    #Protocolos SSL permitidos
    ssl_protocols TLSv1.2 TLSv1.3;

    #Ciphers SSL permitidos a usar por ordem de preferência
```

```

ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:ECDHE-RSA-AES128-GCM-SHA256:AES256+EECDH:DHE-RSA-AES128-GCM-SHA256:
AES256+EDH:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:DHE-
RSA-AES256-SHA:DHE-RSA-AES128-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES256-GCM-SHA384:AES128-
GCM-SHA256:AES256-SHA256:AES128-SHA256:AES256-SHA:AES128-SHA:DES-CBC3-SHA:HIGH:!aNULL:!eNULL:!EXPORT:!
DES:!MD5:!PSK:!RC4";

#DH parameters mais forte
ssl_dhparam $CERTS/dhparam.pem;
#Especifica a curva a usar para os ciphers ECDHE
ssl_ecdh_curve secp384r1;

#Ativa stapling
ssl_stapling on;
ssl_stapling_verify on;

#Melhora tolerância quando o endereço de um domínio muda
resolver 8.8.8.8 8.8.4.4 valid=300s;
resolver_timeout 5s;

#Adição de cabeçalhos que melhoram a segurança
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; preload";
add_header X-DNS-Prefetch-Control off;
add_header X-Frame-Options SAMEORIGIN;
add_header X-Download-Options noopen;
add_header X-Content-Type-Options nosniff;
add_header X-XSS-Protection "1; mode=block";
add_header Content-Security-Policy <Cabeçalho CSP>;
}

```

Exemplo 7.3: Recomendações de segurança na configuração *Nginx*

O valor do cabeçalho `Content-Security-Policy` varia de acordo se for para a [API](#) de dados ou para a interface. A adição dos cabeçalhos que melhoram a segurança na [API](#) de dados é feita pelo `helmet` (Ver [17]) no servidor *Node.js* em vez de ser no *Nginx* mas o resultado final é igual ao uso desta configuração.

Por fim, na configuração do *Nginx* é também adicionado fora dos servers a diretiva

```
ssl_prefer_server_ciphers on;
```

com o objetivo de indicar a preferência de uso dos *ciphers* SSL do servidor definida na configuração (diretiva `ssl_ciphers`) em vez dos *ciphers* do cliente. Ficam assim cumpridas todas as recomendações de segurança e os requisitos apresentados em 7.2, sendo usado para o *deployment* o *docker* e o *docker-compose*. O *deployment* é explicado mais à frente na secção 9.1.

7.4 RESUMO

Adicionalmente aos objetivos desta dissertação foi realizada a migração de [HTTP](#) para [HTTPS](#) da plataforma da [CLAV](#).

Para tal, foi inicialmente investigado o [Certificate Authority \(CA\)](#) gratuito *Let's Encrypt* por forma a usá-lo para obter certificados para ativar o [HTTPS](#) nos websites ([API](#) e interface) da plataforma [CLAV](#).

Na secção da solução apresentou-se os requisitos para esta migração bem como a arquitetura a desenvolver para obter uma plataforma com [HTTPS](#).

Por fim, na secção da implementação descreveu-se os passos realizados, aprofundando a configuração do *Nginx* com várias recomendações de segurança para atingir a arquitetura apresentada na solução cumprindo os requisitos enunciados.

API GATEWAY

A criação de uma *API Gateway* na *CLAV* tem como objetivo principal contrariar o aumento da complexidade da *API* de dados com a adição do controlo de acesso e da autenticação.

Para proceder à criação de uma *API Gateway* é importante primeiro entender o que é e quais os objetivos/aplicações de uma. Além disso, deve-se investigar as várias alternativas existentes no mercado e escolher aquela que melhor se adequa com os nossos requisitos. Na próxima secção são explorados estes pontos.

8.1 ESTADO DA ARTE

8.1.1 *API Gateway*

Uma *API Gateway* encapsula a arquitetura interna em microserviços de uma aplicação, expondo uma única e simples *API* através de um único *URL*, ou seja, um único ponto de entrada para a aplicação. Em arquiteturas baseadas em microserviços o não uso de uma *API Gateway* implica que os utilizadores da aplicação tenham provavelmente de agregar dados de diferentes serviços, manter vários *endpoints*, realizar uma maior quantidade de pedidos e ter possivelmente uma autenticação diferente para cada serviço.

Uma *API Gateway* inclui normalmente [28, 36]:

- Segurança (Autenticação e Autorização)
- Gestão de cotas e *throttling*
- *Caching*
- Processamento e composição da *API*
- Roteamento
- Monitorização da *API*

- Versionamento (possível automatização)
- Balanceamento de carga
- *Rate Limit*

Além disso, a *API Gateway* tem como vantagens [28, 48]:

- Simplifica o código da *API*;
- Oferece uma vista única e central da *API* e, portanto, é mais provável que permita uma política consistente;
- A agregação e transformação de dados simplifica a interação dos clientes com os microserviços distribuídos. A agregação de dados permite também reduzir o número de pedidos;
- Esconde a arquitetura interna e distribui as aplicações baseadas em microserviços reduzindo, geralmente a sobrecarga de configuração;
- Código do cliente mais simples e limpo: quando os serviços cliente e *backend* são separados, o cliente não necessita de saber os vários serviços individuais do *backend* facilitando a manutenção do código bem como a reestruturação dos serviços sem que estas tenham impacto na interação entre cliente e *backend*. Além disso, com uma *API Gateway* não é necessário construir lógica no cliente por forma a acompanhar os *endpoints*;
- Menos latência é igual a uma melhor experiência do utilizador: uma operação do lado do cliente pode necessitar de realizar vários pedidos aos serviços de *backend*, aumentando a latência da operação. Com a presença de uma *API Gateway* pode ser efetuado um único pedido a esta que irá realizar os pedidos internos necessários, agregar os resultados e devolver a resposta ao cliente;
- Autenticação e Encriptação simplificada: Sem o uso de uma *API Gateway* cada serviço de *backend* necessita de tomar as suas decisões de segurança o que aumenta a complexidade do código a desenvolver para um microserviço. Com o aumento da complexidade do código aumenta a possibilidade de erros bem como o aumento da superfície de ataque por utilizadores mal intencionados. Com o uso de uma *API Gateway* toda a segurança está centralizada num único serviço.

Contudo, tem como desvantagens [28]:

- Possível ponto único de falha ou de *bottleneck*;
- Risco de complexidade já que todas as regras da *API* estão num único local;
- Risco de *lock-in* e a migração pode não ser simples.

Olhando um pouco mais da perspectiva da segurança, o controlo de acesso é a principal vantagem de segurança de uma *API Gateway* permitindo a uma organização gerir quem pode aceder à *API* e estabelecer regras de como os pedidos de dados são tratados. O controlo de acesso estende-se também a outras políticas como o *rate limit* às rotas da *API* ou até o pagamento para aceder a certos recursos da *API*.

Como todo o tráfego das rotas da *API* passa por um *gateway*, os especialistas de segurança sentem-se mais confiantes de que têm “no pulso” a *API*. [28]

A *API Gateway* pode introduzir segurança nas mensagens enviadas entre os serviços internos através de encriptação tornando os serviços internos mais seguros. Além disso, é necessário um correto mecanismo de autenticação em conjunto com o uso de *TLS* por forma a evitar o acesso a rotas de pessoas não autorizadas. O uso de um mau mecanismo de autenticação (p.e. ser apenas necessário fornecer o número de telemóvel) pode levar a que qualquer pessoa consiga obter os dados de outra por exemplo.

Outro ponto a ter em conta em relação à proteção de uma *API* é a proteção contra ameaças. Sem esta, a *API Gateway*, a(s) *API(s)* e outros serviços estão inseguros. Ou seja, potenciais hackers ou *malware* podem facilmente tentar propagar ataques tais como *DDoS* ou injeções de *SQL*, *RegExp*, *XML* ou ainda no caso específico da *CLAV* injeções de *SPARQL*. É assim importante realizar validação de *input* da *API*. As validações de *input* mais comuns são:

- Tamanho da mensagem;
- Proteção contra injeções *SQL*;
- Proteção contra *content-level attacks* do *JSON*. Estes ataques são o uso de grandes ficheiros *JSON* por forma a sobrecarregar o *JSON parser* e este eventualmente *crashar*;
- Proteção contra ameaças em *XML*. Estes ataques envolvem normalmente *payloads* recursivos, injeções *SQL* ou *XPath/XSLT* com o mesmo intuito de sobrecarregar o *XML parser* e este eventualmente *crashar*.

Já no caso do tratamento de erros e do código de estado *HTTP* em resposta aos pedidos, é uma boa prática que sejam devolvidos os códigos de estado corretos e com mensagens de erro curtas (apenas o necessário) sem incluírem o *stack trace* visto ser um ponto de insegurança ao permitir qualquer intruso saber por exemplo, as *packages* e as *frameworks* usadas. A *API Gateway* pode ser usada para uniformizar as mensagens de erro devolvidas, impedindo também a possível exposição do código do *backend*.

Por último, ao obrigar a autenticação de todos os utilizadores da *API* e ao manter *logs* dos acessos à *API* torna possível limitar a taxa de consumo da *API* para os utilizadores desta. Muitas *API Gateways* permitem limitar o número de acessos que podem ser feitos para cada recurso da *API* por segundo, minuto, dia ou outra restrição relevante.

Existem várias *API Gateways* das quais se destacam *Express Gateway*, *Kong*, *Moleculer API Gateway*, *Tyk API Gateway* e *Nginx Plus*. Iremos explorar um pouco de cada. Existem outras como, por exemplo,

Amazon's API Gateway, contudo é apenas utilizável se pretendermos usar [AWS](#) e as suas máquinas para o *deployment*. Como não é o caso, não a iremos explorar nesta secção.

Express Gateway

Express Gateway é uma *API Gateway* que pode ser colocada em qualquer arquitetura de microserviços, independentemente da linguagem ou plataforma que se use. O *Express Gateway* protege e expõe os microserviços através de *APIs* usando *Node.js*, *ExpressJS* e *middleware Express*. Além disso é *open-source* e possui as seguintes funcionalidades [32]:

- Políticas empresariais que são normalmente pagas noutras *API Gateways* são aqui gratuitas;
- Configuração através de um ficheiro [YAML](#);
- Arquitetura de *plugins*;
- Extensível com mais de 3000 módulos;
- Corre em qualquer lado (*Docker*, etc);
- Deteta automaticamente e recarrega quando há alterações na configuração;
- Suporta qualquer linguagem e *framework*;
- Suporta todos os casos de uso de microserviços, padrões e arquiteturas;
- Suporta [HTTPS](#), [CORS](#), e [JWT](#) entre outros.

Por definição, o *Express Gateway* usa uma base de dados em memória para testes e para começar a experimentar. O *Express Gateway* pode correr com ou sem *backend* e se uma base de dados persistente for desejada é suportado o *Redis*. A configuração do *Express Gateway* é armazenada num ficheiro [YAML](#) e como tal o *Express Gateway* apenas guarda dados transacionais como informação de utilizadores e de *tokens* de acesso na base de dados. Ou seja, nem sempre é necessário o uso da base de dados, depende do caso de uso.

A configuração como já referido é definida num ficheiro [YAML](#) sendo que existe uma [API](#) e uma [CLI](#) para gerir utilizadores e credenciais. Oficialmente não existe uma [GUI](#) para a [API](#).

Quanto ao desenvolvimento de *plugins* para o *Express Gateway* o mesmo pode ser feito através de *JavaScript* usando a *framework Express*. Os *plugins* são análogos ao *middleware Express*.

Kong

O Kong é uma *API Gateway open-source* escalável, escrita em *Lua* e que pode correr à frente de qualquer *API*. O Kong é construído em cima do *Nginx*, *OpenResty* e *Apache Cassandra* ou *PostgreSQL*. O core do Kong pode ser expandido em termos de funcionalidades e serviços através de *plugins*.

Algumas das funcionalidades presentes são [32]:

- Tarefas de configuração e administração divididas entre *REST API* e *CLI*;
- Extensível através de 36 *plugins* disponíveis (6 deles são comerciais, o resto é *open-source*);
- Corre em qualquer lado (*Kubernetes*, *Docker*, etc);
- Escala ao apenas adicionar mais máquinas;
- Realiza balanceamento de carga dinamicamente através dos vários serviços de *backend*;
- Suporte para um conjunto de políticas para todos os *endpoints* da *API* que pode ser modificado com fluxo condicional;
- Suporta qualquer *framework* e linguagem;
- Suporta todos os casos de uso de microserviços, padrões e arquiteturas;
- Suporta *HTTPS*, *CORS*, e *JWT* entre outros.

A forma mais fácil de instalar (*deploy*) o Kong é através do uso de *Docker* ou de *Kubernetes*.

A configuração do Kong é armazenada no *PostgreSQL* ou no *Cassandra*. Há, contudo, a hipótese de usar uma configuração declarativa que não necessita de uma base de dados para manter a configuração. Esta configuração declarativa é definida pelo utilizador num ficheiro *YAML* ou *JSON* e tem várias vantagens já que reduz a carga de trabalho da máquina de instalação, reduz o número de dependências, diminui a necessidade de manutenção e permite a automatização do processo de *deploy*. [21] A *GUI* oficial de administração do Kong apenas está disponível na versão paga, contudo é possível usar uma versão *third-party* como por exemplo o *Konga*.

Como o Kong é construído sobre o *Nginx* os vários *plugins* necessitam de ser escritos em *Lua*.

Uma das principais desvantagens do Kong é que muitas das funcionalidades têm de ser ativadas através da configuração obrigando a um tempo inicial de *setup*.

Moleculer API Gateway

Moleculer é uma *framework open-source* que contém as funcionalidades mais importantes numa arquitetura baseada em microserviços. Ajuda a construir serviços escaláveis, eficientes e fiáveis oferecendo também várias funcionalidades para construir e gerir os microserviços.

Das várias funcionalidades desta *framework* está presente o módulo da *API Gateway*.

Esta *API Gateway* tem como funcionalidades [39]:

- Suporta [HTTP](#) e [HTTPS](#);
- Serve ficheiros estáticos;
- Suporta *middlewares*;
- Suporta o *upload* de ficheiros;
- Múltiplos *body parsers* ([JSON](#), [urlencoded](#));
- Cabeçalhos [CORS](#);
- [HTTP2](#);
- *Rate limiter*;
- Suporta autorização;
- Modo *middleware*.

Para além disso, esta *API Gateway* pode ser usada como *middleware* numa *API* desenvolvida com *Express* (o caso da *API* da [CLAV](#)). Contudo, o recomendável para usar esta *API Gateway* é que a *API* tenha sido desenvolvida com a *framework Moleculer*.

Tyk API Gateway

O *Tyk* é uma *API Gateway open-source* com componentes gratuitos e outros pagos. Esta plataforma é escrita em *Go* e é constituída por uma *API Gateway* e por um *Dashboard*. Enquanto que o *core* da *API Gateway* é gratuito e *open-source*, o *dashboard* requer a compra de licenças. A versão gratuita permite o uso de uma instância *API Gateway*. Para duas ou mais instâncias é necessário pagar. O *Tyk* também possui uma versão *cloud* (*Tyk Cloud*) em que a versão gratuita permite até 50000 pedidos à *API* diariamente. Acima deste limite é também necessário pagar.

Em termos de funcionalidades, o *Tyk* possui [33]:

- [API RESTful](#);

- Múltiplos protocolos de acesso;
- *Rate limiting* e cotas;
- Controlo de acesso granular;
- Expiração de chaves;
- Versionamento da [API](#);
- *Logs*;
- *Restarts* sem tempo de inatividade;
- Suporte para [HTTPS](#), [CORS](#), [JWT](#) entre outros.

O *core* da *API Gateway* apenas necessita do *Redis* contudo o produto todo (incluindo o *dashboard*) necessita também, como dependências, do *MongoDB* e do *Tyk Pump*. Isto coloca uma maior carga na máquina do servidor que se pode evitar tornando também mais difícil de instalar e de gerir numa máquina local.

Em termos de administração, o *Tyk* possui duas hipóteses, a gestão através da *web GUI* ou através de *REST APIs*. A configuração do *Tyk* é armazenada no *Redis*. O *Tyk* é melhor que os seus concorrentes (*Kong* e *Nginx*) para projetos que pretendem ter a maioria das funcionalidades a funcionar desde o dia um (apenas na versão paga) visto não ser necessário explorar as várias opções disponíveis o que pode demorar algum tempo. [34]

Nginx Plus

O *Nginx Controller* disponível permite gerir o *Nginx Plus* por forma a servir de *load balancer*, *proxy* ou ainda de *API Gateway*. Com o módulo *API Management* do *Nginx Controller* é possível definir, publicar, proteger, monitorizar e analisar *APIs*. Para tal o *Nginx Controller* gera a configuração para o *Nginx Plus*.

Numa comparação de performance com o *Kong* verifica-se que o *Nginx Plus* escala melhor do que o *Kong*. [38] Nos testes realizados chegou-se à conclusão que o módulo *API Management* do *Nginx Plus* adiciona 20% a 30% menos latência aos pedidos dos utilizadores se compararmos com o *Kong*. Além disso, usa menos 40% de *CPU* se compararmos com o *Kong* para a mesma carga de trabalho.

Em termos de funcionalidades o *API Management* do *Nginx Controller* possui [27]:

- Definição e publicação da *API*: Permite criar múltiplas definições de *API* e os seus recursos de componentes, gerir servidores *backend*, direcionar recursos e publicar a configuração resultante nas instâncias *Nginx Plus*;

- *Rate limiting*: Permite mitigar ataques de [DDoS](#) e protege as aplicações de serem inundadas de pedidos ao definir limites de *bandwith* e de pedidos;
- *Autenticação*: Permite autenticar pedidos usando *API keys* e *JWTs*;
- Monitorização e alerta: Permite analisar a performance e as métricas das instâncias *Nginx Plus*. Permite também definir alertas quando alguma métrica ultrapassa certo valor.

Além destas funcionalidades é possível também usar o *Nginx Controller* para criar/gerir *load balancers*. O *Nginx Plus* é uma versão paga e a versão gratuita do *Nginx* não possui o *Nginx Controller* mas é possível usar a versão gratuita através do uso de *plugins* por forma a produzir uma *API Gateway* [34].

O uso da versão gratuita obriga claro a um maior trabalho e a uma metodologia *try and error* por forma a atingir o objetivo final.

A principal dificuldade na utilização do *Nginx* é que a configuração pode ser um pouco complicada de usar e de perceber na sua totalidade possuindo uma acentuada curva de aprendizagem.

8.2 SOLUÇÃO

Nesta secção é escolhida a tecnologia *API Gateway* a usar de acordo com os requisitos necessários. Apresentam-se de seguida os requisitos que a *API Gateway* deve possuir:

- Ser gratuita e *open-source*;
- Suportar [HTTPS](#), [CORS](#) e [JWT](#);
- Suportar Autenticação e Autorização personalizada (Criação de *plugin/middleware*);
- Ser possível de automatizar ao definir apenas ficheiros de configuração sem qualquer uso de [CLIs](#), [GUIs](#) ou [APIs](#);
- Suportar *deployment* em *Docker*;
- Suportar *rate limit* e *cache*;
- Suportar versionamento da [API](#);
- Balancear carga e servir de *reverse proxy*;
- Gerar *logs* e métricas;
- Ser capaz de integrar a documentação desenvolvida na especificação *OpenAPI*.

Antes de fazer uma comparação entre as várias *API Gateways* exploradas no estado de arte (8.1.1) há uma que é imediatamente descartada, a *Molecular API Gateway*. Esta *API Gateway* tem um melhor caso de uso quando a(s) *API(s)* desenvolvida(s) usam a *framework Molecular*. Como este não é o caso da *CLAV* não teremos em consideração esta *API Gateway* na próxima tabela.

Requisito	<i>Nginx</i>	<i>Kong</i>	<i>Tyk</i>	<i>Express Gateway</i>
Ser gratuita e <i>open-source</i>	✓	✓	✓	✓
Suportar HTTPS, CORS e JWT	JWT apenas na versão paga. Necessário usar <i>plugin</i>	✓	✓	✓
Suportar Autenticação e Autorização personalizada (Criação de <i>plugin/middleware</i>)	✓ (em <i>Lua</i>)	✓ (em <i>Lua</i> ou <i>Go</i>)	✓ (em <i>Python, Lua</i> ou <i>Javascript</i>)	✓ (em <i>Javascript</i>)
Ser possível de automatizar ao definir apenas ficheiros de configuração sem qualquer uso de CLIs, GUIs ou APIs	✓	✓	✗	✓
Suportar deployment em <i>Docker</i>	✓	✓	✓	✓
Suportar <i>rate limit</i> e <i>cache</i>	✓	✓	✓	Não possui <i>cache</i>
Suportar versionamento da API	✗	✗	✓	✗
Balancear carga e servir de <i>reverse proxy</i>	✓	✓	✓	✓
Gerar <i>logs</i> e métricas	✓	✓	✓	✓
Ser capaz de integrar a documentação desenvolvida na especificação <i>OpenAPI</i>	✗	✗	✓	✗

Tabela 4: Comparação entre *API Gateways* [34, 32, 33]

Infelizmente nenhuma das tecnologias suporta todos os requisitos. Assim a partir da tabela 4 irá ser escolhida a tecnologia a usar tendo em conta a importância de cada requisito.

Das quatro tecnologias presentes na tabela, podemos desde já descartar o *Tyk* visto não permitir automatizar o *deployment*. Sobram, assim, o *Nginx*, o *Kong* e o *Express Gateway*. Destas três, verifica-se que a última possui menos funcionalidades e portanto não será a tecnologia escolhida.

Entre o *Nginx* e o *Kong* a escolha irá prender-se com a facilidade e poder de configuração visto que os requisitos presentes em cada tecnologia são muito semelhantes. O *Kong* é construído a partir do *Nginx* (daí as suas parecenças em termos de requisitos presentes) e é possível configurar o *Nginx* a partir do *Kong*. Para além disso, o *Kong* possui vários *plugins* por forma a permitir certas funcionalidades que no *Nginx* seria necessário configurar manualmente. Portanto a escolha tecnológica para a *API Gateway* recai sobre o *Kong*.

8.2.1 Arquitetura

Após a escolha da *API Gateway* a usar (*Kong*) é necessário proceder ao esboço de uma arquitetura base a desenvolver. Esta arquitetura apresenta-se de seguida, na figura 19, onde apenas se inclui a *API* visto a interface não sofrer qualquer alteração:

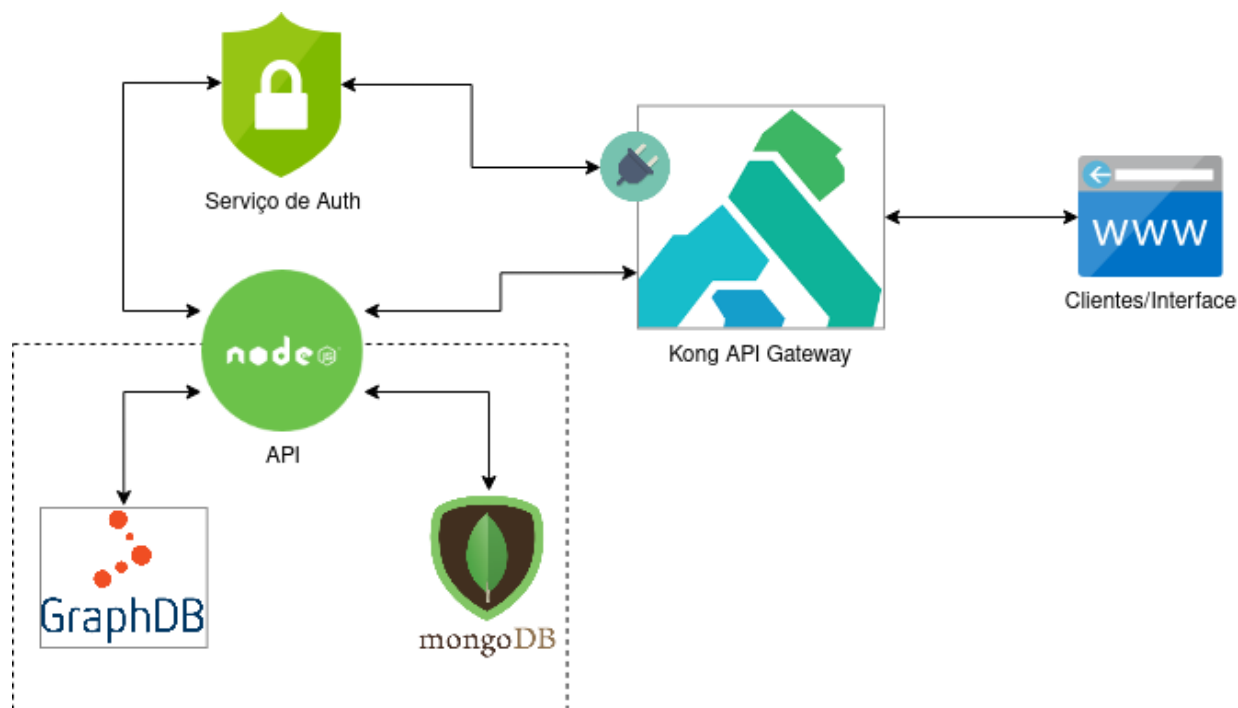


Figura 19: Arquitetura a desenvolver com *API Gateway*

Apesar das várias vantagens do *Kong* não é possível, através dos *plugins* fornecidos por este, realizar uma proteção das rotas semelhante à presente na *API* de dados da *CLAV*. Portanto, com o intuito de simplificar a *API* de dados, a ideia passa por retirar da *API* de dados da *CLAV* tudo que envolva a proteção desta, tais como a verificação de *tokens* (autenticação), a geração de *tokens* bem como a verificação da autorização de um utilizador (verificar se o nível do utilizador é suficiente para aceder determinada rota), e colocar estas funcionalidades noutra servidor independente da *API* de dados. Este servidor está representado na arquitetura como Serviço de *Auth* e procederá então à autenticação e autorização dos pedidos realizados pelos clientes.

Com esta separação a *API* de dados precisará de realizar pedidos ao Serviço de *Auth* para a verificação e geração de *tokens* em casos particulares como o *login* de um utilizador ou a *criação* de uma Chave *API*.

Todos os pedidos realizados pelos clientes e pela interface da *CLAV* têm como ponto de entrada o *Kong*. Este, através de um *plugin*, para cada pedido recebido irá realizar um pedido ao Serviço de *Auth* onde:

1. Verifica quem pode aceder (que tipo de clientes) à rota do pedido;

2. Procede à autenticação do pedido (se necessário);
3. Procede à autorização do pedido (se necessário).

Caso a resposta do Serviço de *Auth* seja positiva então o *Kong* procederá à realização do pedido à *API* de dados. Caso contrário, o pedido à *API* de dados não é efetuado e é devolvida à máquina cliente uma resposta com o erro correspondente (erro de autenticação ou de autorização).

8.3 IMPLEMENTAÇÃO

Nesta secção irá ser abordada a configuração do *Kong* além do desenvolvimento do Serviço de *Auth* adicional para a proteção da *API* de dados.

8.3.1 Serviço de *Auth*

Este serviço, desenvolvido em *Node.js*, tem como principal papel a autenticação e autorização dos pedidos recebidos na plataforma *CLAV*. Além disso, este serviço irá tratar de verificar e gerar *tokens* (*JWTs*).

Para saber quem pode aceder determinada rota, o Serviço de *Auth* possui um dicionário em que cada chave é um verbo *HTTP* e em cada verbo existe um dicionário com os vários caminhos desse verbo. A cada caminho é associado um valor que indica quem pode aceder a rota. Este valor é:

- **-1:** Todos podem aceder
- **0:** Só podem aceder Chaves *API* autenticadas ou utilizadores autenticados
- **Um número maior que 0:** Só podem aceder utilizadores autenticados com nível igual ou superior ao número
- **Lista de números maiores que 0:** Só podem aceder utilizadores autenticados com nível presente nesta lista

A ordem dos caminhos em cada verbo é importante visto que as rotas são testadas por ordem e, quando há um *match*, é assumido que é essa a rota do pedido não sendo testado o resto dos caminhos.

Consoante o valor obtido do dicionário é feita a autenticação e autorização necessária para a rota. Caso não seja obtido um valor, assume-se que a rota não existe na *API* de dados.

Sempre que for acrescentada uma nova rota na *API* de dados é necessário adicionar essa neste dicionário com as devidas permissões.

Este serviço de autenticação e autorização disponibiliza cinco rotas:

POST /AUTH Autenticar e Autorizar um pedido. Recebe no *body* o verbo, o caminho, a *query string* e os cabeçalhos do pedido a efetuar para a [API](#) de dados. Como resposta devolve respostas com *HTTP status*:

- 200: o pedido pode ser efetuado à [API](#) de dados. Esta resposta possui no *body* a informação contida no *token* se este for enviado no pedido;
- 401: *token* inválido ou pedido sem autenticação para aceder a rota;
- 403: pedido sem autorização para aceder a rota;
- 404: a rota não existe.

POST /USER/SIGN Gerar um *token* para um utilizador. Recebe no *body* do pedido a informação do utilizador e o tempo de expiração a usar para o *token*. Como resposta devolve o *token* gerado;

POST /USER/VERIFY Verificar um *token* de um utilizador. Recebe no *body* o *token* do utilizador. Como resposta, caso o *token* seja válido, devolve a informação contida no *token*;

POST /APIKEY/SIGN Gerar um *token* para uma Chave API. Recebe no *body* do pedido a informação da Chave API e o tempo de expiração a usar para o *token*. Como resposta devolve o *token* gerado;

POST /APIKEY/VERIFY Verificar um *token* de uma Chave API. Recebe no *body* o *token* da Chave API. Como resposta, caso o *token* seja válido, devolve a informação contida no *token*.

Nenhuma destas rotas está disponível externamente, ou seja, é apenas acessível pelo *Kong* e pela [API](#) de dados.

A necessidade de rotas diferentes para utilizadores e Chaves API deve-se ao facto de serem usados pares de chave pública/privada diferentes para gerar e verificar os *tokens*. Para esta geração e verificação é usada a mesma biblioteca usada até agora, `jsonwebtoken`. Continua-se também a usar os extratores da biblioteca `passport-jwt` para obter os *tokens* dos pedidos, mas já não se recorre ao `passport` para proceder à autenticação visto não ser necessário.

Com a existência deste serviço, a [API](#) de dados sem *Kong* deixou de estar protegida e nos casos em que necessita, de forma excepcional, de verificar ou de gerar *tokens* recorre à [API](#) do serviço de *Auth*. Apesar disso continua a ser essencial que a [API](#) de dados saiba quem realizou o pedido. Essa informação será enviada num cabeçalho pelo *Kong* algo que será explicado na próxima secção. Ou seja, o serviço de *Auth* não é apenas um *middleware* na *pipeline* do *Kong* mas um serviço na rede de microserviços da plataforma [CLAV](#).

8.3.2 *Plugin external-auth*

Como já referido anteriormente não é possível, usando os *plugins* disponíveis do *Kong*, obter uma proteção semelhante à presente na *API* de dados. Para resolver esta situação havia duas hipóteses, criar um *plugin* que tratasse da autenticação e da autorização ou criar um *plugin* que interceptasse os pedidos, realizasse um pedido a um serviço externo para tratar da autenticação e da autorização, e consoante a resposta deixasse ou não o pedido ser realizado. A segunda hipótese foi a escolhida visto que implica a criação de um *plugin* de menores dimensões, que tem de ser desenvolvido em *Lua*, e permite por outro lado aproveitar parte do código de autenticação e autorização presente na *API* de dados e usá-lo no serviço externo. Ou seja, é uma abordagem mais rápida e simples para além de que este *plugin* de menores dimensões encontrava-se já em parte desenvolvido¹ onde foi apenas necessário realizar algumas melhorias como a possibilidade de devolver respostas 403 e 404 entre outras alterações².

De uma forma mais pormenorizada, quando um pedido é recebido pelo *Kong*, este pedido poderá passar por vários *plugins* de acordo com a configuração usada antes e após ser efetuado o pedido à *API* de dados. O *plugin external-auth* se adicionado na configuração é executado antes do pedido ser efetuado à *API* de dados. Este *plugin* obtém do pedido o caminho, o verbo, a *query string* e os cabeçalhos e envia-os no *body* de um pedido que faz ao serviço externo. Quando o *plugin* recebe a resposta do serviço externo, consoante o *HTTP status* da resposta, irá ter a seguinte reação:

- 200: Insere no pedido a efetuar à *API* de dados o cabeçalho *CLAV-Auth* onde é enviada em formato *string* a resposta do serviço externo;
- 401: Devolve uma resposta de erro para o utilizador com *HTTP status* 401;
- 403: Devolve uma resposta de erro para o utilizador com *HTTP status* 403;
- 404: Devolve uma resposta de erro para o utilizador com *HTTP status* 404;
- Restantes: Devolve uma resposta de erro para o utilizador com *HTTP status* 401.

Portanto quando um pedido possui autenticação e autorização pelo serviço externo, este pedido inclui também um cabeçalho *CLAV-Auth* onde estará presente a informação presente no *token*, seja de um utilizador ou de uma Chave API. Este cabeçalho será interpretado pela *API* de dados por forma a esta saber quem realizou o pedido. No caso do cabeçalho não ser enviado, for vazio ou num formato incorreto, a *API* de dados irá considerar que o pedido foi efetuado por alguém “Desconhecido”.

Este *plugin* está disponível em <https://luarocks.org/modules/jcm300/external-auth>.

¹Ver <https://github.com/aunkenlabs/kong-external-auth>

²Ver <https://github.com/jcm300/kong-external-auth>

8.3.3 Configuração Kong

O *Kong* pode ser configurado através de um ficheiro de configuração `.conf` onde é definido o modo em que executa, possíveis ligações a bases de dados, configurações do *Nginx*, entre outros. Contudo não é neste ficheiro que se define o comportamento que o *Kong* efetua quando recebe determinado pedido, apesar de se poder alterar algum comportamento através de configurações do *Nginx*. Para definir o comportamento do *Kong* há três hipóteses. A que foi escolhida, configuração declarativa, permite que o *Kong* execute sem necessitar de uma base de dados para armazenar as configurações visto que as configurações são definidas num ficheiro [YAML](#) ou [JSON](#) que é carregado para memória. Assim não se sobrecarrega desnecessariamente o servidor que executa o *Kong*.

Para usar esta configuração declarativa é necessário no ficheiro de configuração `.conf` colocar a diretiva `database` com o valor `off` e a diretiva `declarative_config` com o caminho do ficheiro da configuração declarativa.

Na configuração declarativa são usados vários *plugins* a maioria desenvolvidos pelo *Kong* exceto o *external-auth*.

Por predefinição os *plugins* do *Kong* estão disponíveis. Para se poder usar *plugins* externos é necessário indicá-los no ficheiro `.conf` na diretiva `plugins`. Assim o valor aqui colocado foi `blunded,external-auth` indicando que se pretende usar os *plugins* do *Kong* (`blunded`) e o *external-auth*.

Antes de iniciar o *Kong* será depois necessário instalar o *plugin external-auth*.

No ficheiro `.conf` é também indicado para onde os logs de erro e acesso são enviados sendo os de erro enviados para o `stderr` e os de acesso para o `stdout`. Além disso indica-se também em que portas ([HTTP](#) e [HTTPS](#)) o *Kong* irá receber os pedidos bem como em que portas está acessível a [API](#) de administrador (como é usada uma configuração declarativa esta [API](#) será apenas de leitura).

O primeiro passo na configuração declarativa foi adicionar o serviço da [API](#) de dados e associar desde logo o *plugin* necessário para a proteção da [API](#) de dados:

```
services:
- name: API
  url: ${API_HOST}
  routes:
  - name: TodasRotas
    paths:
    - /
  plugins:
  - name: external-auth
    config:
      url: ${SERVER_AUTH_HOST}
      path: /auth
```

Exemplo 8.1: Configuração declarativa do *Kong*: [API](#) de dados

Com esta configuração inicial o *Kong* reencaminha todos os pedidos que comecem por / para a *API* de dados, usando neste serviço o *plugin* `external-auth` já descrito. Temos assim uma primeira versão funcional com autenticação e autorização da *API* de dados.

Contudo a *API* de dados necessita de permitir `CORS` e para tal é adicionado o *plugin* do *Kong* chamado `cors`[23]:

```
services:
- name: API
  ...
  plugins:
  ...
  - name: cors
    service: API
    config:
      origins:
      - '*'
      methods:
      - GET
      ...
      headers:
      - Accept
      ...
    credentials: true
```

Exemplo 8.2: Configuração declarativa do *Kong*: *plugin* `cors`

Além deste foram adicionados mais 3 *plugins* do *Kong* a este serviço, cada um com objetivos diferentes.

O *plugin* `rate-limiting` por forma a limitar o número de pedidos efetuados a 10 pedidos por segundo por endereço IP[25]:

```
services:
- name: API
  ...
  plugins:
  ...
  - name: rate-limiting
    config:
      second: 10
      policy: local
```

Exemplo 8.3: Configuração declarativa do *Kong*: *plugin* `rate-limiting`

É possível também aplicar este *plugin* a rotas específicas da *API* de dados.

Por outro lado, o *plugin* `proxy-cache` é usado para realizar a *cache* de respostas de pedidos `text/plain` (texto) ou `application/json` (*JSON*) com verbos `GET` e `HEAD` e *HTTP status* 200, 301 e 404 por uma hora[24]:

```
services:
```

```
- name: API
...
plugins:
...
- name: proxy-cache
  config:
    cache_ttl: 3600 #segundos (1h) em cache
    strategy: memory
```

Exemplo 8.4: Configuração declarativa do *Kong*. *plugin* proxy-cache

Com este *plugin* é possível evitar pedidos recentes à *API* de dados iguais bem como acelerar o tempo de resposta dos pedidos que foram recentemente efetuados.

É ainda usado o *plugin* response-transformer por forma a adicionar cabeçalhos na resposta devolvida pela *API* de dados. Estes cabeçalhos têm como objetivo melhorar a segurança da *API* e cumprir algumas das recomendações de segurança *HTTPS* já referidas em 7.2 [26]:

```
services:
- name: API
...
plugins:
...
- name: response-transformer
  config:
    remove:
      headers:
        - strict-transport-security
        ...
        - content-security-policy
    add:
      headers:
        - 'Strict-Transport-Security: max-age=31536000; includeSubDomains; preload'
        ...
        - "Content-Security-Policy: default-src 'none'"
```

Exemplo 8.5: Configuração declarativa do *Kong*. *plugin* response-transformer

Este *plugin* possui a seguinte ordem de execução: remover (*remove*), renomear (*rename*), substituir (*replace*), adicionar (*add*) e acrescentar (*append*).

A adição dos *plugins* (cors e response-transformer) permite assim a remoção da *API* de dados das bibliotecas cors e helmet que eram usadas para alcançar os mesmos objetivos para os quais estes *plugins* vão ser usados.

A adição do cabeçalho Content-Security-Policy com o valor default-src 'none' no *plugin* response-transformer impede qualquer conteúdo que não a resposta em si. Contudo a *API* de dados possui uma página de documentação na rota /<Versão da API>/docs que irá necessitar de um Content-Security-Policy diferente.

Para tal, definiu-se uma rota na configuração declarativa associando-a ao serviço da [API](#) de dados mas no qual o *plugin* `response-transformer` possui um valor diferente para o cabeçalho `Content-Security-Policy`. Os outros *plugins* associados ao serviço da [API](#) de dados continuam a ser executados para esta rota mas o *plugin* `response-transformer` desse serviço é substituído por este definido na rota:

```
routes:
- name: docs
  service: API
  strip_path: false
  paths:
  - /$API_VERSION/docs
  methods:
  - GET
  plugins:
  - name: response-transformer
    config:
      remove:
        headers:
        - strict-transport-security
        ...
        - content-security-policy
      add:
        headers:
        - 'Strict-Transport-Security: max-age=31536000; includeSubDomains; preload'
        ...
        - "Content-Security-Policy: default-src 'self' $DOMAINS; img-src 'self' https://validator.swagger.io data: $DOMAINS; style-src 'self' 'unsafe-inline' $DOMAINS; script-src 'self' 'unsafe-inline' $DOMAINS"
```

Exemplo 8.6: Configuração declarativa do *Kong*. Rota da documentação

A propriedade `strip_path` com valor falso impede que o *Kong* associe (parecido com um *bind*) o caminho `/<Versão API>/docs` do *Kong* ao caminho `/` da [API](#) de dados. Assim, quando se faz o pedido a esta rota o pedido que é feito na [API](#) de dados é no caminho `/<Versão API>/docs` desta. Por fim, procedeu-se à configuração do [HTTPS](#) no *Kong*. Como se usam os certificados do *Let's Encrypt* pode-se usar o *plugin* `acme` do *Kong*[22]:

```
plugins:
- name: acme
  config:
    account_email: $EMAIL
    domains: $DOMAINS_LIST
    renew_threshold_days: 30
    storage: redis
    storage_config:
      redis:
        auth: "redisPass123"
        port: 6379
        database: 0
```

```
host: $REDIS_HOST
tos_accepted: true
```

Exemplo 8.7: Configuração declarativa do *Kong*: *plugin* acme

Para manter a configuração e os certificados do *plugin* após reinícios do *Kong* usa-se o *Redis* para guardar a configuração bem como os certificados. Convém referir que este *plugin* tem de ser global na configuração declarativa e não associado a um único serviço. Além disso, para permitir a geração dos certificados pelo *Let's Encrypt* é necessário permitir o acesso à rota que começa em `/.well-known/acme-challenge` para permitir a validação do controlo do domínio. Assim foi adicionado o seguinte serviço à configuração declarativa:

```
services:
- name: acme-dummy
  url: http://127.0.0.1:65535
  routes:
  - name: acme-dummy
    paths:
    - /.well-known/acme-challenge
- name: API
...
```

Exemplo 8.8: Configuração declarativa do *Kong*: Serviço para a geração de certificados [TLS](#)

Para ativar também o uso do *plugin* acme é necessário no ficheiro `.conf` indicar onde estão os certificados confiáveis por forma a ser possível o *Kong* iniciar para depois poder obter os certificados do *Let's Encrypt*. Para tal, é indicado na diretiva `lua_ssl_trusted_certificate` o valor `/etc/ssl/certs/ca-certificates.crt` (esta é a localização no caso de uma distribuição *Ubuntu*, poderá ser diferente noutras distribuições e *OS's*). Quando o *Kong* iniciar, para acionar a criação do certificado, é necessário fazer um pedido ao *Kong* correndo o seguinte comando:

```
curl https://<domain> -k
```

substituindo `<domain>` pelo domínio da [API](#) de dados.

Além disso, por forma a que quando alguém tente aceder às rotas através de [HTTP](#) seja redirecionado para o [HTTPS](#) foram adicionadas as seguintes propriedades nas rotas:

```
services:
...
- name: API
...
  routes:
  - name: TodasRotas
    protocols:
    - https
    https_redirect_status_code: 301
```

```

...
...
routes:
- name: docs
  ...
  protocols:
  - https
  https_redirect_status_code: 301
  ...

```

Exemplo 8.9: Configuração declarativa do *Kong*: Serviço para a geração de certificados TLS

A propriedade `protocols` indica que protocolos deverão estar disponíveis na rota e a propriedade `https_redirect_status_code` indica o *status code* nos casos em que os pedidos são efetuados para o HTTP. Como este *status code* é 301 os clientes serão redirecionados para o HTTPS.

Por fim, por forma a melhorar a segurança do HTTPS e cumprir as recomendações de segurança são adicionadas no ficheiro de configuração `.conf` algumas diretivas de configuração do *Nginx*:

```

#Indicar que se pretende usar ciphers personalizados
ssl_cipher_suite = custom
#Ciphers permitidos a usar por ordem de preferência
ssl_ciphers = EECDH+AESGCM:EDH+AESGCM:ECDHE-RSA-AES128-GCM-SHA256:AES256+EECDH:DHE-RSA-AES128-GCM-SHA256:AES256+
EDH:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:
ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA:DHE
-RSA-AES128-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-
SHA256:AES128-SHA256:AES256-SHA:AES128-SHA:DES-CBC3-SHA:HIGH:!aNULL:!eNULL:!EXPORT:!DES:!MD5:!PSK:!RC4

#Protocolos SSL permitidos
nginx_http_ssl_protocols = TLSv1.2 TLSv1.3
#Indicar que se deve preferir os ciphers do servidor em vez do cliente
nginx_http_ssl_prefer_server_ciphers = on

#DH parameters mais forte
nginx_http_ssl_dhparam = /etc/ssl/certs/dhparam.pem

#Especifica a curva a usar para os ciphers ECDHE
nginx_http_ssl_ecdh_curve = secp384r1

#Ativar stapling, infelizmente não funciona porque o plugin acme usado ainda não suporta
nginx_http_ssl_stapling = on
nginx_http_ssl_stapling_verify = on

#Melhorar tolerância quando o endereço de um domínio muda
nginx_http_resolver = 8.8.8.8 8.8.4.4 valid=300s
nginx_http_resolver_timeout = 5s

#Tempo em que pode ser reusada uma sessão
nginx_http_ssl_session_timeout = 1d

```



```
#Tempo em que é mantida uma conexão idle a um servidor
nginx_upstream_keepalive_timeout = 65
```

Exemplo 8.10: Configurações do *Nginx* no ficheiro de configuração `.conf`

Uma pequena nota. As variáveis ambiente (`${var}` ou `$var`) serão substituídas antes de serem usadas pelo *Kong* ao aplicar o comando `envsubst` ao ficheiro da configuração declarativa.

Convém também referir que, esta versão da *API* de dados com *Kong* respeita todos os requisitos enunciados na migração de *HTTP* para *HTTPS* (7.2) com o objetivo primordial de melhorar a segurança. Para a *deployment* foi também usado o *docker* e o *docker-compose* o que é descrito na secção 9.2.

8.3.4 Arquitetura

Na figura 19 apresentou-se uma visão inicial da arquitetura. Já na figura 20 apresenta-se a arquitetura final com o *Kong* e sem incluir a interface da *CLAV*:

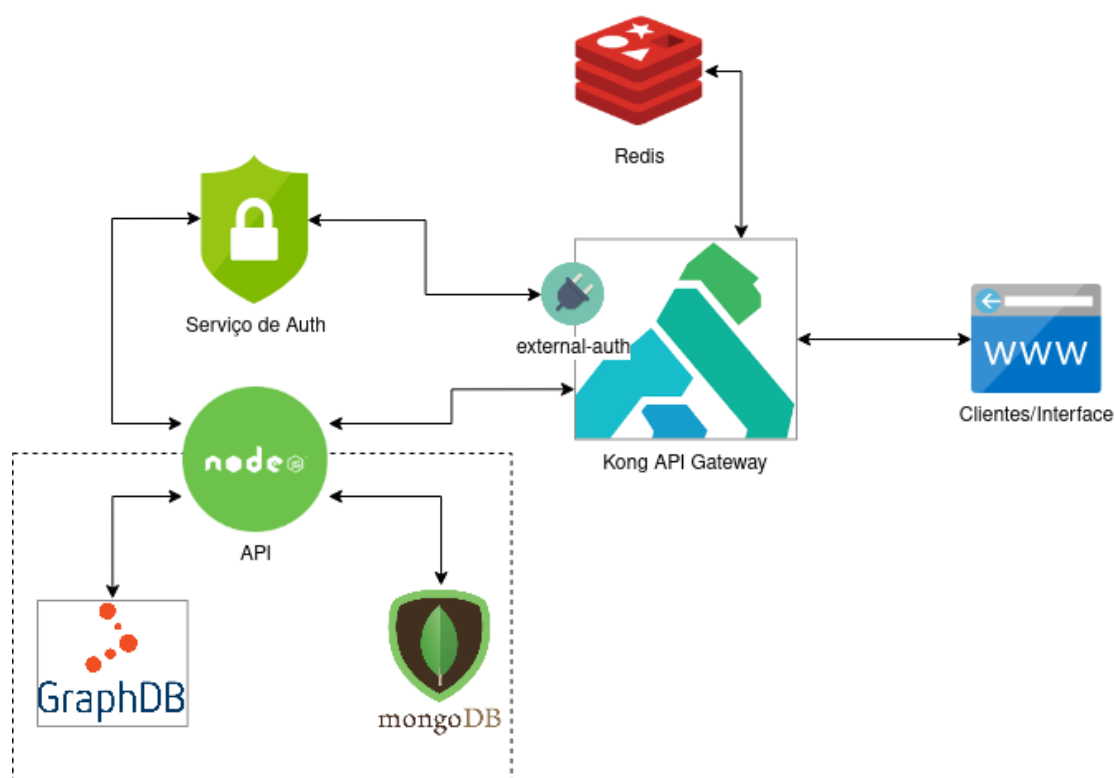


Figura 20: Arquitetura desenvolvida com *API Gateway*

8.4 RESUMO

De forma resumida, na secção do estado da arte foi descrito o que é uma *API Gateway* e seus objetivos bem como várias alternativas para a sua implementação existentes no mercado.

Já na secção da solução são descritos os requisitos necessários na plataforma *CLAV*, escolhendo a alternativa que melhor se adequa.

A criação de uma *API Gateway* na *CLAV* tem como principal intuito a simplificação da autenticação/proteção da *API* de dados pelo que são considerados os requisitos mais apropriados a esta situação.

Finalmente, na secção da implementação, descrevem-se os passos realizados (desenvolvimento e configuração) para implementar a *API Gateway* e apresenta-se a arquitetura final desenvolvida.

DEPLOYMENT

No decorrer desta dissertação foram também desenvolvidos os mecanismos de *deployment* necessários tanto para a versão sem *Kong* bem como para a versão com *Kong*. Para facilitar a instalação e a manutenção do código, criaram-se vários “*containers*” cada um contendo um serviço. Mais tarde, orquestraram-se dois meta-serviços utilizando o `docker-compose`, a API de dados e a interface. Neste capítulo descrevem-se os procedimentos de *deployment* e de manutenção de cada versão.

9.1 VERSÃO SEM *KONG*

9.1.1 Primeira instalação

Para uma primeira instalação deve-se garantir que a máquina de instalação tem instalado o `git`, o `docker` e o `docker-compose`. A especificação do `docker-compose` está na versão 3.5 pelo que é necessário a versão 17.06.0 ou superior do `docker` e a versão 1.18.0 ou superior do `docker-compose`.

A instalação do *software* depende do OS pelo que numa máquina *Ubuntu* o *software* pode ser instalado através das seguintes linhas de comando [11, 10]:

```
sudo apt-get update
sudo apt-get install -y git

sudo apt-get install -y apt-transport-https ca-certificates curl gnupg-agent software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io

sudo curl -L \
  "https://github.com/docker/compose/releases/download/1.25.5/docker-compose-$(uname -s)-$(uname -m)" \
  -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

Exemplo 9.1: Instalar *docker* e *docker-compose*

O passo seguinte é abrir as portas na máquina de instalação que serão usadas para a API dados e/ou interface (É necessário abrir duas portas para a API de dados (HTTP e HTTPS) e duas para a interface (HTTP e HTTPS)). Se instalar a API de dados e a interface na mesma máquina as 4 portas têm de ser diferentes.

A seguir, é necessário realizar a clonagem dos repositórios git que se vão instalar. Posicionamo-nos na raiz da pasta respetiva e mudamos para o *branch* da versão que queremos instalar:

API de dados:

```
git clone https://github.com/jcramalho/CLAV2018.git
cd CLAV2018
git checkout https
```

Interface:

```
git clone https://github.com/jcramalho/CLAV2019.git
cd CLAV2019
git checkout https
```

Numa versão de produção, é de extrema importância realizar ainda os seguintes passos referentes à configuração da autenticação da CLAV (tanto na API de dados como na interface):

Na API de dados gerar dois pares de chaves pública/-privada:

```
#assume-se que se encontra na pasta CLAV2018
openssl genrsa -out config/keys/apiKey 2048
openssl rsa -in config/keys/apiKey -pubout \
  -out config/keys/apiKey.pub
openssl genrsa -out config/keys/userKey 2048
openssl rsa -in config/keys/userKey -pubout \
  -out config/keys/userKey.pub
```

Na interface copiar para esta as chaves públicas geradas (este comando apenas funciona se a API e a interface estão na mesma máquina):

```
#assume-se que se encontra na pasta CLAV2018
cp config/keys/apiKey.pub config/keys/userKey.pub \
  <raiz da pasta da interface>/src/plugins/keys/
```

Sem estes passos qualquer utilizador que saiba as chaves privadas usadas pode gerar um *token* e assim ter acesso indevido à CLAV. Portanto as chaves privadas devem estar apenas acessíveis na máquina de produção da API de dados (não partilhe nem faça *push* no repositório Git destas chaves).

Antes de avançar posicionamo-nos na pasta *deploy* (`cd deploy`) para a realização dos próximos passos.

Criação de Imagens

As imagens usadas na instalação podem ser geradas com antecedência, fora da máquina de instalação. Para isso essas imagens geradas devem ser registadas no *DockerHub* para posterior uso (*push*).

API de dados

É necessário criar duas imagens: *graphdb* e *server*.

Para criar a imagem *graphdb* são necessários dois ficheiros adicionais: um ficheiro zip com a distribuição do *GraphDB* e um ficheiro *Turtle* com a ontologia a carregar para o *GraphDB*.

A distribuição do *GraphDB* deve ser obtida através do site do *GraphDB*¹ ao preencher o formulário. Após o preenchimento do formulário irá receber um email do *GraphDB*, como na figura 21, de onde deve realizar o *download* da versão “stand-alone server”. O ficheiro zip tem um nome parecido com “graphdb-free-8.11.0-dist.zip” onde “free-8.11.0” indica a versão do *GraphDB* que deve colocar na variável ambiente GRAPHDB_VERSION do ficheiro *.env* da pasta *deploy*.

Deve colocar o ficheiro zip na pasta *graphdb* da pasta *deploy*. Nesta pasta (*graphdb*) deve também colocar o ficheiro *Turtle* e indicar na variável ambiente GRAPHDB_DATA_FILE o nome deste ficheiro (ex: *clav-2020-01-04-false.ttl*). Sempre que um destes ficheiros é alterado é necessário criar de novo a imagem.

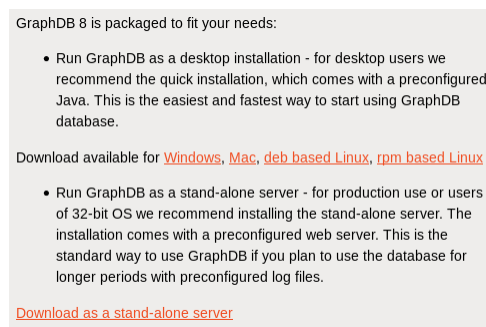


Figura 21: Parte do email recebido pelo *GraphDB*

A criação da imagem *server* é necessária quando não existe ou quando o código da *API* de dados muda.

Interface

Existe apenas uma imagem (*interface*) que deve ser recriada sempre que:

- a versão da *API* de dados (variável ambiente *API_VERSION* do ficheiro *.env* da pasta *deploy*) muda;
- o *URL* da *API* de dados para onde a interface irá fazer os pedidos (variável ambiente *API_URL* do ficheiro *.env* da pasta *deploy*) muda;
- a versão da interface (variável ambiente *INTERFACE_VERSION* do ficheiro *.env* da pasta *deploy*) muda;
- o código muda.

Pode associar uma *tag* a cada imagem a criar, no ficheiro *.env* da *API* de dados, nas variáveis ambiente GRAPHDB_IMG (imagem *graphdb*) e SERVER_IMG (imagem *server*) onde deve inserir o nome do

¹Versão gratuita: <https://www.ontotext.com/products/graphdb/graphdb-free/>

repositório do *DockerHub* e a *tag* usar (<repositório>:<tag>). Já no ficheiro *.env* da interface está presente a variável ambiente *INTERFACE_IMG* com o mesmo intuito mas para a imagem da interface. Para criar estas imagens basta correr na pasta *deploy* da *API* de dados (para criar as imagens da *API* de dados) ou da interface (para criar a imagem da interface):

```
docker-compose -f docker-compose-build.yml build
```

Depois de criadas as imagens se pretender torná-las disponíveis no *DockerHub* deve para cada uma delas realizar o seguinte comando:

```
docker push <repositório>:<tag>
```

Configuração

Tanto na *API* de dados como na interface na pasta *deploy* está presente um ficheiro *.env* já referido. Estes ficheiros permitem configurar respetivamente a instalação da *API* de dados e da interface.

Na *API* de dados para além das já referidas variáveis ambiente (*GRAPHDB_VERSION*, *GRAPHDB_DATA_FILE*, *GRAPHDB_IMG* e *SERVER_IMG*) possui também as seguintes variáveis ambiente:

Variável Ambiente	Descrição
<i>CERT_FOLDER</i>	Pasta onde estão os ficheiros do certificado no <i>container</i> <i>nginx</i>
<i>ACME_FOLDER</i>	Pasta onde estão os ficheiros do <i>acme.sh</i> no <i>container</i> <i>nginx</i>
<i>API_VERSION</i>	Versão da <i>API</i> de dados
<i>DOMAINS</i>	Domínios dos quais a <i>API</i> de dados irá estar acessível
<i>SWAGGER_URL</i>	URL principal a colocar na documentação <i>OpenAPI</i> (<i>Swagger</i>)
<i>INTERFACE_HOSTS</i>	<i>Hosts</i> das interfaces da <i>CLAV</i> que podem aceder a <i>API</i> de dados
<i>HTTP_PORT</i>	Porta HTTP em que a <i>API</i> de dados recebe os pedidos HTTP dentro da máquina de instalação
<i>HTTPS_PORT</i>	Porta HTTPS em que a <i>API</i> de dados recebe os pedidos HTTPS dentro da máquina de instalação

Tabela 5: Variáveis ambiente do ficheiro *.env* da *API* de dados sem *Kong*

Quanto à interface para além das já referidas variáveis ambiente (*API_VERSION*, *INTERFACE_VERSION*, *API_URL* e *INTERFACE_IMG*) possui também as seguintes variáveis ambiente:

Variável Ambiente	Descrição
CERT_FOLDER	Pasta onde estão os ficheiros do certificado no <i>container interface</i>
ACME_FOLDER	Pasta onde estão os ficheiros do <code>acme.sh</code> no <i>container interface</i>
DOMAINS	Domínios dos quais a interface irá estar acessível
SERVER_URL	URL da API de dados
HTTP_PORT	Porta HTTP em que a interface recebe os pedidos HTTP dentro da máquina de instalação
HTTPS_PORT	Porta HTTPS em que a interface recebe os pedidos HTTPS dentro da máquina de instalação
NGINX_FILE	Ficheiro de configuração <i>Nginx</i> a usar. Duas hipóteses: <ul style="list-style-type: none"> <code>nginx.conf.template</code>: A interface não faz redirecionamento dos pedidos que recebe para a API de dados. O valor de <code>API_URL</code> tem de ser o URL real da API de dados (será igual a <code>SERVER_URL</code>) <code>nginxProxy.conf.template</code>: A interface faz redirecionamento dos pedidos que recebe para a API de dados. O valor de <code>API_URL</code> tem de ser o URL da interface, estando em <code>SERVER_URL</code> o URL real da API de dados

Tabela 6: Variáveis ambiente do ficheiro `.env` da interface

Tanto na API de dados como na interface, caso sejam alteradas estas variáveis ambiente não é necessário recriar as imagens, basta apenas reiniciar os *containers*.

Após realizar as alterações necessárias pode correr o seguinte comando na pasta `deploy` da API de dados (para iniciar a API de dados) ou da interface (para iniciar a interface):

```
docker-compose up
```

Este comando também funciona mesmo que as imagens não estejam criadas desde que estejam presentes no *DockerHub* (verifica de acordo com os valores das variáveis ambiente `GRAPHDB_IMG` e `SERVER_IMG` na API e `INTERFACE_IMG` na interface).

Caso as imagens (`server` e `graphdb` ou `interface`) ainda não estejam criadas (nem presentes no *DockerHub*) pode criá-las e de seguida iniciá-las através do comando:

```
docker-compose -f docker-compose-build.yml up
```

Na API de dados se pretende manter os dados entre atualizações (criação de novas imagens e reinícios dos *containers*) não deve eliminar os volumes `clav-mongodb-data` (dados do BD *MongoDB*) e `clav-graphdb-data` (dados do BD *GraphDB*).

Além disso, não deve eliminar os volumes `acme-data` e `crontabs`, o primeiro para não ser necessário gerar novos certificados (podendo atingir o limite de geração de certificados do *Let's Encrypt*) e o segundo visto ser o ficheiro de configuração que permite que execute periodicamente (diariamente) o `acme.sh` para verificar se os certificados têm apenas 30 dias de validade e como tal proceder à renovação destes.

Do lado da interface os volumes que não deve eliminar são `acme-interface-data` e `crontabs-interface` pelas mesmas razões já referidas para os volumes `acme-data` e `crontabs`.

Povoamento do MongoDB

Até ao momento, nesta instalação, a `BD MongoDB` continua vazia. Se já possui uma `BD` do `MongoDB` já povoada é possível proceder à migração. Caso contrário precisará de pelo menos inserir à mão (através do cliente do `MongoDB`) um utilizador com o nível de Administrador de Perfil Tecnológico à coleção `users` caso contrário será impossível registar/inserir utilizadores pela interface já que apenas pessoal registado e com determinado nível pode registar/inserir utilizadores.

Backup

De forma a realizar a migração é necessário primeiro fazer o backup da `BD` já povoada. Para realizar o backup executa-se o comando seguinte:

```
mongodump --db <nome da BD> --out <caminho a guardar o backup>
```

No destino final será criada uma pasta com o nome da `BD` que irá possuir o backup da `BD`. Para se poder realizar backup é necessário que o `MongoDB` esteja a correr.

Importação da informação

Primeiro, é necessário parar os `containers`:

```
docker stop clav_nginx
docker stop clav_server
docker stop clav_mongo clav_graphdb
```

Para importar os dados nesta instalação basta executar o seguinte comando:

```
docker run --rm -v clav-mongodb-data:/data/db \
-v <caminho absoluto onde foi guardado o backup>/<nome da BD>:/backup mongo \
bash -c "(mongod &) && mongorestore --db <nome da BD> --drop /backup"
```

No fim, é necessário voltar a iniciar os `containers`:

```
docker start clav_mongo clav_graphdb
docker start clav_server
docker start clav_nginx
```

Melhorias de segurança do HTTPS

Após a conclusão da instalação, recomenda-se a realização de duas melhorias de segurança do `HTTPS`. Uma delas apenas se pode realizar neste momento.

Nesta secção quando se refere o domínio ou domínios pretende-se fazer referência aos presentes nos DOMAINS dos ficheiros de configuração `.env` da API de dados e da interface.

A melhoria apenas agora possível é a adição dos domínios à *HSTS preload list*. Para isso deve aceder a <https://hstspreload.org/>, começar por inserir um dos domínios e seguir os passos indicados. Repita para os restantes domínios. O *HSTS preloading* permite que os *browsers* saibam à partida que domínios devem ser acedidos apenas por HTTPS.²

A outra melhoria de segurança é a adição de *CAA records* associados aos domínios no DNS (tem de suportar *CAA records*) no qual os domínios estão indexados. Os *CAA records* a adicionar podem ser obtidos em <https://sslmate.com/caa/> indicando o nome do domínio e escolhendo como *Authorized Certificate Authorities* o *Let's Encrypt*. Na secção 4 da página Web estão presentes os *CAA records* a adicionar. Repita para os restantes domínios. Estes *CAA records* indicam a(s) autoridade(s) emissora(s) de certificados permitida(s) para o(s) domínio(s) por forma a impedir a emissão de um certificado para o(s) mesmo(s) domínio(s) noutra autoridade emissora de certificados.³

9.1.2 Atualizações

Tal como a instalação, as atualizações podem ser feitas de forma separada, ou seja, é possível atualizar apenas a API de dados ou apenas a interface ou até as duas se assim se pretender.

Para tal, deve-se primeiro parar os *containers*:

API de dados:

```
docker stop clav_nginx
docker stop clav_server
docker stop clav_mongo clav_graphdb
```

Interface:

```
docker stop interface
```

Antes de recriar as imagens se pretender colocá-las no *DockerHub* deve alterar as *tags* nos ficheiros de configuração `.env` da(s) imagem(s) que precisará de recriar por forma a não sobrescrever *tags*. Fica à escolha do utilizador. Apenas tem de ter em conta que as variáveis ambiente GRAPHDB_IMG, SERVER_IMG e INTERFACE_IMG podem ser utilizadas para gerir as imagens da forma que pretender.

De seguida recriam-se os *containers* necessários:

- Caso tenha mudado a ontologia ou pretenda atualizar a distribuição do *GraphDB* recria-se a imagem do `clav_graphdb` após as devidas alterações no ficheiro `.env` da pasta `deploy`.

```
docker-compose -f docker-compose-build.yml build graphdb
```

No caso do ficheiro da ontologia ter sido alterado é necessário correr também:

²Ver <https://scotthelme.co.uk/hsts-preloading/> para perceber esta melhoria de segurança

³Ver <https://sslmate.com/caa/about> para perceber esta melhoria de segurança

```
docker rm clav_graphdb
docker volume rm clav-graphdb-data
```

- Caso tenha sido alterado código na API de dados (`git pull`) recria-se a imagem do `clav_server`:

```
git pull #obter novos commits
docker-compose -f docker-compose-build.yml build server
```

- Caso tenha sido alterado código da interface (`git pull`) ou uma das variáveis ambientes `API_URL`, `INTERFACE_VERSION` e/ou `API_VERSION` tenham sido alteradas recria-se a imagem da interface:

```
git pull #obter novos commits
docker-compose -f docker-compose-build.yml build interface
```

Caso o `build` use a `cache` para construir as imagens adicione a `flag` `--no-cache` no comando de `build` (a seguir à palavra `build`).

Por fim, inicia-se os `containers` da API de dados ou da interface dependendo em que pasta `deploy` está através do comando:

```
docker-compose up -d
```

Note que, a opção `-d` é necessária para o comando libertar novamente a consola e colocar os `containers` a correr em `background`.

9.1.3 Backup

Em termos de backup, os volumes de maior importância são o do `MongoDB` e o do `GraphDB` visto possuírem toda a informação da `CLAV`. Note que este backup apenas implica a API de dados. Por forma a realizar o backup deve:

```
#parar containers
docker stop clav_nginx
docker stop clav_server
docker stop clav_mongo clav_graphdb

#backup do volume do GraphDB
docker run --rm --volumes-from clav_graphdb \
  -v $(pwd):/backup ubuntu \
  bash -c "cd /opt/graphdb/home/data/repositories && \
  tar cvf /backup/clav_graphdb.tar ."
```

```
#backup do volume do MongoDB
docker run --rm --volumes-from clav_mongo \
  -v $(pwd):/backup ubuntu bash -c "cd /data/db && \
  tar cvf /backup/clav_mongo.tar ."
```

```
#Pode agora iniciar os containers
docker start clav_mongo clav_graphdb
docker start clav_server
docker start clav_nginx
```

Exemplo 9.2: Backup dos volumes do *docker*

No final possuirá dois ficheiros **TAR** (`clav_graphdb.tar` e `clav_mongo.tar`) na raiz onde executou os comandos que são respetivamente o backup do *GraphDB* e o backup do *MongoDB*.

A partir daqui pode armazenar estes onde achar mais conveniente e seguro.

9.1.4 Migração

Para realizar a migração para outra máquina é necessário primeiro realizar o backup dos volumes da máquina atual. Pode para isso seguir a descrição dada na secção 9.1.3. Depois deve copiar os ficheiros comprimidos (**TAR**) para a nova máquina de instalação.

Segue-se a realização da primeira instalação na nova máquina, bastando para isso seguir a descrição dada na secção 9.1.1 mas sem realizar o povoamento do *MongoDB*.

O passo final é restaurar os volumes através dos backups (ficheiros comprimidos). Para tal siga os seguintes passos para restaurar os volumes:

```
#assumindo que se encontra na raiz onde se encontram os ficheiros TAR

#parar containers
docker stop clav_nginx
docker stop clav_server
docker stop clav_mongo clav_graphdb

#restauro do volume do GraphDB
docker run --rm -v clav-graphdb-data:/opt/graphdb/home/data/repositories \
  -v $(pwd):/backup ubuntu bash -c "cd /opt/graphdb/home/data/repositories \
  && tar xvf /backup/clav_graphdb.tar"

#restauro do volume do MongoDB
docker run --rm -v clav-mongodb-data:/data/db \
  -v $(pwd):/backup ubuntu bash -c "cd /data/db \
  && tar xvf /backup/clav_mongo.tar"

#voltar a iniciar os containers
docker start clav_mongo clav_graphdb
docker start clav_server
docker start clav_nginx
```

Exemplo 9.3: Restauro dos volumes do *docker*

Pode também dividir a API de dados e a interface por duas máquinas. Para isso deve realizar na mesma o backup dos volumes na máquina atual, seguir a primeira instalação individualmente para cada máquina (instalando apenas a componente correspondente), copiar os backups apenas para a máquina da API de dados e restaurar esses backups.

9.2 VERSÃO COM KONG

Nesta versão será apenas abordado a instalação da API de dados já que a adição do Kong não implicou qualquer alteração na interface.

9.2.1 Primeira instalação

Os dois primeiros passos são exatamente iguais aos já descritos na primeira instalação da versão sem Kong (ver 9.1.1), instalar as dependências necessárias e abrir as portas na máquina de instalação para a API de dados.

De seguida realiza-se a clonagem do repositório git, muda-se de raiz e de branch:

```
git clone https://github.com/jcm300/docker-clav.git
cd docker-clav
git checkout kong
```

Além disso, obtém-se o conteúdo dos submódulos do repositório ao correr:

```
git submodule update --init
```

Tal como na versão sem Kong, numa versão de produção é necessário gerar dois novos pares de chaves pública/privada e copiar as novas chaves públicas para a interface:

Gerar dois pares de chaves pública/privada:

```
#assume-se que se encontra na pasta docker-clav
openssl genrsa -out CLAV-auth/config/keys/apiKey 2048
openssl rsa -in CLAV-auth/config/keys/apiKey -pubout \
  -out CLAV-auth/config/keys/apiKey.pub
openssl genrsa -out CLAV-auth/config/keys/userKey 2048
openssl rsa -in CLAV-auth/config/keys/userKey -pubout \
  -out CLAV-auth/config/keys/userKey.pub
```

Na interface copiar para esta as chaves públicas geradas (este comando apenas funciona se a API e a interface estão na mesma máquina):

```
#assume-se que se encontra na pasta docker-clav
cp CLAV-auth/config/keys/apiKey.pub \
  CLAV-auth/config/keys/userKey.pub \
  <raiz da pasta da interface>/src/plugins/keys/
```

Criação de Imagens

A criação da imagem graphdb não sofre muitas alterações em relação à versão sem Kong (ver 9.1.1). Algumas das diferenças a ter em conta são que a pasta graphdb e o ficheiro .env encontram-se na pasta docker-clav. A criação da imagem server não sofreu quaisquer alterações.

A maior diferença desta versão em relação à versão sem *Kong* é a necessidade de também criar a imagem `clav-auth` que, como a imagem `server`, apenas necessita de ser criada quando não existe ou quando o seu código muda.

Configuração

No ficheiro `.env` desta versão para além das variáveis ambiente `GRAPHDB_VERSION`, `GRAPHDB_DATA_FILE`, `GRAPHDB_IMG` e `SERVER_IMG` com o mesmo intuito da versão sem *Kong* há as seguintes variáveis ambiente:

Variável Ambiente	Descrição
<code>SERVER_AUTH_IMG</code>	Tem a mesma finalidade que a variável <code>SERVER_IMG</code> mas para o servidor de autenticação
<code>EMAIL</code>	Email a usar para gerar os certificados
<code>API_VERSION</code>	Versão da <i>API</i> de dados
<code>DOMAINS</code>	Domínios dos quais a <i>API</i> de dados irá estar acessível
<code>INTERFACE_HOSTS</code>	<i>Hosts</i> das interfaces da <i>CLAV</i> que podem aceder
<code>HTTP_PORT</code>	Porta <i>HTTP</i> em que a <i>API</i> de dados recebe os pedidos <i>HTTP</i> dentro da máquina de instalação
<code>HTTPS_PORT</code>	Porta <i>HTTPS</i> em que a <i>API</i> de dados recebe os pedidos <i>HTTPS</i> dentro da máquina de instalação

Tabela 7: Variáveis ambiente do ficheiro `.env` da *API* de dados com *Kong*

Caso sejam alteradas estas variáveis ambiente (tirando `SERVER_AUTH_IMG`) não é necessário recriar as imagens, basta apenas reiniciar os *containers*.

Após realizar as alterações necessárias, para criar as imagens e iniciar a *API* de dados basta correr:

```
docker-compose -f docker-compose-build.yml up
```

Caso as imagens já estejam criadas ou estejam presentes no *DockerHub* pode correr:

```
docker-compose up
```

De igual forma como na versão sem *Kong* não deve eliminar os volumes `clav-mongodb-data` (dados da *BD MongoDB*), `clav-graphdb-data` (dados da *BD GraphDB*) e, além dessas, não deve eliminar `clav-acme-data` (dados referentes aos certificados e à configuração destes).

Povoamento do MongoDB

O povoamento do *MongoDB* pode ser efetuado de igual forma como na primeira instalação sem *Kong* (ver 9.1.1). As únicas diferenças é que para parar os *containers* deve executar:

```
docker stop clav_kong
docker stop clav_redis clav_auth clav_server
docker stop clav_mongo clav_graphdb
```

e para iniciar os *containers* deve correr:

```
docker start clav_mongo clav_graphdb
docker start clav_redis clav_auth clav_server
docker start clav_kong
```

Melhorias de segurança do HTTPS

Quanto às melhorias de segurança são iguais e são aplicadas de igual forma como na versão sem *Kong* (ver 9.1.1).

9.2.2 Atualizações

Numa atualização com *Kong* um dos primeiros passos é verificar se há atualizações:

```
#assumindo que se encontra na pasta docker-clav
git pull #obter novos commits
```

De resto as únicas diferenças em relação à versão sem *Kong* (ver 9.1.2) são o código a executar para parar os *containers*. Além disso, tal como a imagem do `clav_server`, quando o código do serviço de autenticação muda é necessário recriar a imagem do `clav-auth`:

```
#assumindo que se encontra na pasta docker-clav
cd CLAV-auth
git checkout master
git pull #obter novos commits
cd ..
docker-compose -f docker-compose-build.yml build clav-auth
```

Por fim, outra diferença é que para recriar a imagem do `clav_server` deve executar:

```
#assumindo que se encontra na pasta docker-clav
cd CLAV2018
git checkout kong
git pull #obter novos commits
cd ..
docker-compose -f docker-compose-build.yml build server
```

9.2.3 Backup

No backup as únicas diferenças em relação à versão sem *Kong* (ver 9.1.3) são o código a executar para parar e para iniciar os *containers*.

9.2.4 Migração

Na migração será seguir a primeira instalação desta versão. De resto, os passos serão iguais aos apresentados na versão sem Kong (ver 9.1.4) tirando o código para parar e para iniciar os *containers*.

9.2.5 Forçar renovação do certificado HTTPS

Caso necessite de forçar a renovação do certificado HTTPS nesta versão com Kong deve primeiro apagar a configuração do HTTPS presente no Redis:

```
# Aceder o Redis
docker exec -it clav_redis redis-cli
#Autenticar
auth redisPass123
# Ver todas as entradas no Redis
keys *
# Apagar todas as entradas no Redis
del <key1> <key2> ...
# Sair
exit
```

E depois deve reiniciar o Kong, onde este irá obter o novo certificado do *Let's Encrypt*.

```
# Reiniciar Kong
docker restart clav_kong
```

Esta renovação só funciona caso tenha os *containers* a correr.

9.3 VERSÃO SEM KONG VS COM KONG

Esta secção tem como objetivo realizar uma comparação entre as duas versões com o objetivo de decidir qual delas se adequa mais.

Para realizar os testes nesta secção, cada componente foi executado numa máquina do *Digital Ocean* com *Ubuntu 20.04 x64*, 8 GB de memória RAM e 4 CPUs. Para traduzir o domínio no IP da máquina foi usado o *No-IP*⁴.

Para comparar a performance foi realizado um teste em que 100 utilizadores fazem dois pedidos, um pedido das classes e quando termina um pedido de uma classe específica. Um utilizador começa os seus pedidos 1 segundo após o anterior. Para realizar este teste foi usado o *Apache JMeter*. Os resultados obtidos foram os seguintes (atenção que a *cache* estava “quente” nas duas versões, portanto o pedido das classes deverá ser proveniente da cache):

⁴ver <https://www.noip.com/>

Versão API	Pedido	Tempo de Resposta (média em ms)	% de Erro	Throughput (pedidos por)
sem Kong	Classes	1650	0	59.9/min
	Classe	465	0	1.0/s
	Total	1057	0	2.0/s
com Kong	Classes	2469	0	59.6/min
	Classe	588	0	1.0/s
	Total	1529	0	2.0/s

Tabela 8: Resultados de performance para a API de dados

Olhando para os resultados, o *throughput* é bastante semelhante entre as duas, contudo o tempo de resposta é menor na versão sem Kong, o que faz algum sentido já que a versão com Kong (o Kong tem embutido o Nginx) é mais pesada em termos de componentes em execução.

Em termos de facilidade de instalação, as duas são bastante semelhantes pelo que quem conseguir instalar com uma delas consegue instalar a outra.

Quanto à proteção da API apesar das várias alterações (componente externo à API de dados), são semelhantes tendo como principal diferença o não uso do passport na versão com Kong o que não afeta a forma como é realizada a proteção. De certa forma, o uso do passport na versão sem Kong é atualmente desnecessária desde que se coloque em req.user a informação presente no token do utilizador. A essência da proteção da API de dados manteve-se basicamente igual.

9.3.1 Configuração do HTTPS

A plataforma *SSL Labs* possui uma ferramenta *online* que permite-nos auferir a qualidade da configuração do HTTPS num domínio.⁵ Para perceber melhor a classificação realizada pelo *SSL Labs* recomenda-se a leitura de <https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide>.

Nesta secção foi também comparada a configuração do HTTPS da interface para além das versões da API:

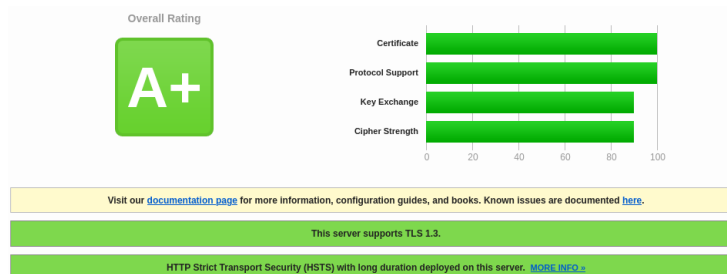


Figura 22: Classificação SSL Labs da API sem Kong

⁵Ver <https://www.ssllabs.com/ssltest/>

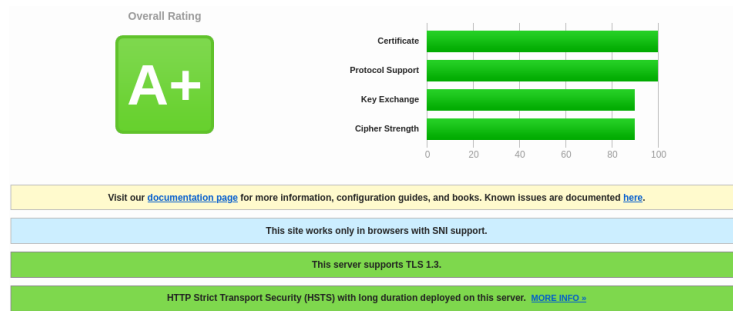


Figura 23: Classificação SSL Labs da API com Kong

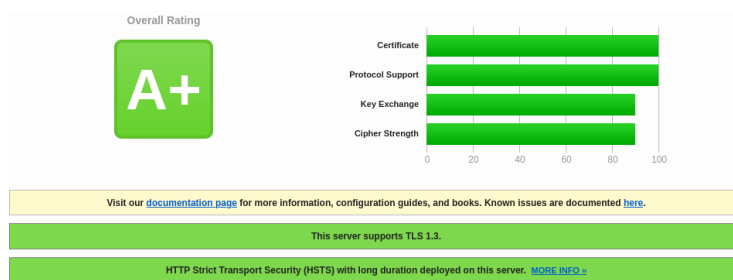


Figura 24: Classificação SSL Labs da Interface

Como é possível verificar pelas figuras 22, 23 e 24 as 3 tem a classificação máxima, A+, bem como a mesma classificação em cada componente específica. A API sem Kong é normal ter a mesma classificação que a interface porque a configuração usada é praticamente a mesma. Quanto ao caso da API de dados com Kong esta apenas funciona em *browsers* que suportam SNI sendo raros aqueles que não suportam atualmente. Além disso a versão com Kong não tem ativado o OCSP stapling (uma das recomendações de segurança) enquanto que nos outros casos esta está ativa. Estas duas diferenças da versão com Kong devem-se à forma como o Kong e o plugin do Kong acme funcionam.

Por fim, nos 3 casos há um único sinal de aviso, que se deve à falta do CAA record (uma das melhorias de segurança referidas) no DNS. Contudo, com os resultados obtidos conseguimos ter alguma certeza de que as configurações realizadas são atualmente seguras.

9.3.2 Resumo

Em conclusão, apesar de a configuração do HTTPS da API com Kong ser ligeiramente pior bem como a performance (tempo de resposta), em termos gerais permite obter futuramente uma melhor API, com uma maior modularidade. Permite também futuramente escalar mais facilmente a API de dados bem como modularizar esta API em mais componentes.

9.4 RESUMO

Neste capítulo foi descrito o procedimento de instalação da [API](#) sem *Kong*, da [API](#) com *Kong* e da interface. No fim é feita uma comparação entre as duas versões da [API](#), concluindo que no nosso caso se adequa mais a versão com *Kong*.

CONCLUSÕES E TRABALHO FUTURO

Esta dissertação tem vários objetivos distintos mas todos com a motivação de continuar o desenvolvimento da plataforma **CLAV** cumprindo alguns dos requisitos em falta na plataforma.

A documentação em *OpenAPI* (nova versão do *Swagger*) da **API** de dados da **CLAV** desenvolvida está presente em <https://clav-api.dglab.gov.pt/v2/docs/> e facilita assim o desenvolvimento da plataforma **CLAV** bem como quem desenvolve sistemas arquivísticos que irão usar esta **API** visto a documentação possuir uma descrição em cada rota, os parâmetros que se podem usar bem como exemplos e as estruturas dos *bodies* e das respostas.

A exportação na **API** de dados da **CLAV** dos principais tipos de dados em **XML**, **CSV** e **RDF** foi também alcançada, tendo ainda sido desenvolvido adicionalmente uma interface de exportação que permite escolher o *encoding* de saída. Permite assim que os utilizadores possam, por exemplo visualizar/transformar os dados em *Excel* ou ferramentas similares para além de permitir que aplicações possam usar outros formatos para tratar estes dados, por exemplo, ao exportar em **XML**.

Por outro lado, foi protegida a **API** de dados com múltiplos níveis de acesso, garantindo assim que os dados presentes na **API** da **CLAV** não são alterados indevidamente por utilizadores mal intencionados. Assim, apenas pessoal autorizado (de acordo com o nível de utilizador) pode fazer alterações.

Além disso, é agora também possível entrar na plataforma **CLAV** através de **Chave Móvel Digital (CMD)**, um dos mecanismos de autenticação disponíveis no **Autenticação.gov**. É dada assim mais uma hipótese de autenticação aos utilizadores para além das já implementadas através de **CC** e de *email+password*.

Através do *Kong* foi desenvolvida uma **API Gateway** separando da **API** de dados toda a proteção, colocando esta num serviço independente. O uso do *Kong* permitirá futuramente se assim se pretender modularizar ainda mais a **API** de dados em microserviços. Da perspetiva dos utilizadores a proteção da **API** de dados manteve-se igual, mas permitiu simplificar a complexidade do código da **API** de dados.

Quanto ao objetivo de integração da **CLAV** no **iAP** este é conseguido com o uso do **Autenticação.gov** para autenticar os utilizadores a quem é requisitado a obtenção de dados pessoais do utilizador, como o **NIC** e o nome completo para autenticar o utilizador na **CLAV**.

São assim atingidos todos os objetivos da dissertação. Adicionalmente realizou-se a migração de **HTTP** para **HTTPS** melhorando a segurança de utilização da plataforma para além da exigência desta para

se poder utilizar a versão de produção do Autenticação.gov. Além disso, apesar de não referidas neste documento, já ele extenso, foram adicionados mecanismos de *cache* na API de dados, otimizado o cálculo do fecho transitivo da LC, desenvolvido a importação de uma TS a partir de um ficheiro, desenvolvida uma interface de pesquisa avançada para a LC e uma interface para o registo de utilizadores para uma entidade, adicionada a filtragem das classes da LC de acordo com o nível do utilizador e com o *status* de cada classe e foi adicionado a validação de parâmetros na API de dados com recurso ao *express-validator* garantindo uma maior resiliência da API e devolvendo um maior *feedback* aos utilizadores indicando que parâmetros estão errados e porquê. Foi também adicionada a capacidade de guardar os *logs* dos pedidos efetuados à API, sua consulta e sua gestão e vários mecanismos de gestão da plataforma, como alteração de parâmetros, exportar e apagar coleções bem como a gestão da cache.

Quanto às questões de investigação introduzidas no início desta dissertação, a produção da documentação da API de dados é realizada através de múltiplos ficheiros por forma a modularizar a documentação. Estes ficheiros estão organizados estruturalmente de acordo com o seu papel. Por forma, a gerar o documento final com a documentação em *OpenAPI* é usado o *yaml-include*. O documento final é por fim usado pelo *SwaggerUI*, através do *swagger-ui-express*, para produzir uma documentação dinâmica.

Relativamente à *API gateway* esta permite várias vantagens numa arquitetura em microserviços, desde oferecer uma vista única (único ponto de entrada) da API, permitir esconder a arquitetura interna bem como simplificar a autenticação e autorização da API já que há só uma entrada a proteger (a superfície de ataque é menor se compararmos com uma arquitetura em microserviços em que cada microserviço tem a sua própria autenticação e autorização).

Por fim, a autonomização do serviço de autenticação traz como benefícios:

- os pedidos sem autenticação não chegam agora à API de dados reduzindo a carga sobre esta;
- o serviço de autenticação, sendo este um microserviço, pode ser colocado numa máquina à parte;
- simplificação do código da API de dados, melhorando a facilidade de manutenção deste;
- maior facilidade na manutenção da autenticação da API de dados, por exemplo na alteração das permissões de acesso de determinada rota.

10.1 TRABALHO FUTURO

Foram alcançados todos os objetivos desta dissertação, contudo certas tarefas requerem uma manutenção constante. Uma delas é a documentação em *OpenAPI* que necessita de ser atualizada sempre que são adicionadas novas rotas à API ou quando alguma das rotas presentes mude, seja na resposta, nos parâmetros, na funcionalidade ou ainda caso esta passe a *deprecated*. Outra tarefa que requer manutenção é a exportação para *CSV*. Se se pretender que mais alguma rota possua exportação para *CSV* ou caso

alguma das respostas das rotas que possuem exportação para *CSV* sejam alteradas é necessário adicionar ou alterar, respetivamente, no conversor de *JSON* para *CSV* como deve ser feita a transformação dos objetos *JSON* de saída da rota. Há ainda outra tarefa de manutenção e esta deve também ser realizada sempre que é adicionada uma nova rota. Esta tarefa consiste em adicionar ao serviço de *Auth*, na versão com *Kong*, quem pode aceder a essa rota, ao adicionar as permissões dessa rota no dicionário de permissões das rotas presente nesse serviço.

Já quanto à continuação do trabalho desenvolvido por esta dissertação destaca-se a *API* de dados com *Kong*. Apesar do que foi desenvolvido estar num estado passível de ser usado em produção, há varias melhorias que podem ser desenvolvidas desde replicar a *API* de dados colocando o *Kong* a servir de *load balacing* até colocar *rate limiting* especificamente para cada rota com o objetivo principal de melhorar a performance e a tolerância a falhas da *API* de dados.

Ainda a partir da versão da *API* de dados com *Kong*, outra questão de investigação que se pode abordar é uma maior modularização da *API* de dados, investigando a performance comparativamente com a versão atual bem como o uso de *message-brokers* (como *RabbitMQ* e *Kafka*) para a troca de informação entre os vários microserviços.

BIBLIOGRAFIA

- [1] Alives. HOWTO: A+ with all 100%'s on SSL Labs test using apache2.4 (READ WARNINGS), 10 2015. URL <https://community.letsencrypt.org/t/howto-a-with-all-100-s-on-ssl-labs-test-using-apache2-4-read-warnings/2436>. Acedido a 2020-06-12.
- [2] AMA. *Autenticação.gov - Fornecedor de autenticação da Administração Pública Portuguesa*, 1.5.1 edition, 12 2018.
- [3] AMA. *Autenticação.gov*, 2019. URL <https://autenticacao.gov.pt/fa/Default.aspx>. Acedido a 2019-11-20.
- [4] AMA. *Autenticação.gov - Fornecedor de autenticação da Administração Pública Portuguesa*, 1.5.3 edition, 02 2020.
- [5] Auth0. Introduction to JSON Web Tokens, 2019. URL <https://jwt.io/introduction/>. Acedido a 2019-12-19.
- [6] Filipa Carvalho, Helena Neves, Rita Gago, and Alexandra Lourenço. Aplicação de uma tabela de seleção, 2016. URL http://arquivos.dglab.gov.pt/wp-content/uploads/sites/16/2017/08/FT5_Aplicacao-TS.pdf. Acedido a 2020-06-01.
- [7] Maria José Chaves and Alexandra Lourenço. Elaboração de Relatórios de Avaliação de Documentação Acumulada, 12 2016. URL http://arquivos.dglab.gov.pt/wp-content/uploads/sites/16/2017/08/FT1_RADA.pdf. Acedido a 2020-06-03.
- [8] Cloudflare. Types of SSL Certificates | SSL Certificate Types Explained, 2020. URL <https://www.cloudflare.com/learning/ssl/types-of-ssl-certificates/>. Acedido a 2020-06-03.
- [9] DGLAB. CLAV - Classificação e Avaliação da Informação Pública, 2019. URL <http://clav.dglab.gov.pt>. Acedido a 2019-12-15.
- [10] Docker. Install Docker Compose, 2020. URL <https://docs.docker.com/compose/install/>. Acedido a 2020-06-12.
- [11] Docker. Install Docker Engine on Ubuntu, 2020. URL <https://docs.docker.com/engine/install/ubuntu/>. Acedido a 2020-06-12.

- [12] Let's Encrypt. How It Works, 10 2019. URL <https://letsencrypt.org/how-it-works/>. Acedido a 2020-06-04.
- [13] Let's Encrypt. Rate Limits, 3 2020. URL <https://letsencrypt.org/docs/rate-limits/>. Acedido a 2020-06-06.
- [14] Let's Encrypt. Getting Started, 2020. URL <https://letsencrypt.org/getting-started/>. Acedido a 2020-06-03.
- [15] Express. Production Best Practices: Security, 2017. URL <https://expressjs.com/en/advanced/best-practice-security.html>. Acedido a 2020-06-12.
- [16] Zélia Gomes and Alexandra Lourenço. Boas práticas de eliminação de documentos, 2019. URL http://arquivos.dglab.gov.pt/wp-content/uploads/sites/16/2019/08/FT9_Eliminacao_2019-08-20.pdf.pdf. Acedido a 2020-06-01.
- [17] Helmet. Docs, 2020. URL <https://helmetjs.github.io/docs/>. Acedido a 2020-06-12.
- [18] Helmet. Content Security Policy, 2020. URL <https://helmetjs.github.io/docs/csp/>. Acedido a 2020-06-12.
- [19] Pekka Henttonen and Kimmo Kettunen. Functional classification of records and organisational structure. *Records Management Journal*, 21:86–103, 07 2011. doi: 10.1108/09565691111152035.
- [20] Foundeo Inc. Content Security Policy Reference, 2020. URL <https://content-security-policy.com/>. Acedido a 2020-06-12.
- [21] Kong. DB-less and Declarative Configuration, 2020. URL <https://docs.konghq.com/2.0.x/db-less-and-declarative-config/>. Acedido a 2020-05-16.
- [22] Kong. ACME, 2020. URL <https://docs.konghq.com/hub/kong-inc/acme/>. Acedido a 2020-05-29.
- [23] Kong. CORS, 2020. URL <https://docs.konghq.com/hub/kong-inc/cors/>. Acedido a 2020-05-28.
- [24] Kong. Proxy Cache, 2020. URL <https://docs.konghq.com/hub/kong-inc/proxy-cache/>. Acedido a 2020-05-28.
- [25] Kong. Rate Limiting, 2020. URL <https://docs.konghq.com/hub/kong-inc/rate-limiting/>. Acedido a 2020-05-28.

- [26] Kong. Response Transformer, 2020. URL <https://docs.konghq.com/hub/kong-inc/response-transformer/>. Acedido a 2020-05-28.
- [27] Karthik Krishnaswamy. Introducing NGINX API Management: Manage NGINX Plus API Gateways with NGINX Controller, 10 2018. URL <https://www.nginx.com/blog/introducing-nginx-api-management-api-gateways-with-nginx-controller/>. Acedido a 2020-05-14.
- [28] Guy Levin. The Role of API Gateways in API Security, 8 2018. URL <https://dzone.com/articles/the-role-of-api-gateways-in-api-security>. Acedido a 2020-05-11.
- [29] Hal Lockhart, Thomas Wisniewski, Prateek Mishra, and Nick Ragouzis. *Security Assertion Markup Language(SAML) V2.0 Technical Overview*. OASIS, 7 2005.
- [30] Alexandra Lourenço. Aplicação do destino final: O papel do dono e do participante, 2019. URL http://arquivos.dglab.gov.pt/wp-content/uploads/sites/16/2019/08/FT8_Aplica%C3%A7%C3%A3o-DF-Dono-Participante_2019-07-25.pdf. Acedido a 2020-06-01.
- [31] Alexandra Lourenço, José Carlos Ramalho, Maria Rita Gago, and Pedro Penteadó. Plataforma CLAV: contributo para a disponibilização de dados abertos da Administração Pública em Portugal. Acedido a 2019-11-20, 7 2019. URL http://eprints.rclis.org/38643/1/Plantilla_EDICIC2019Barcelona-PT-COM_Penteadó_v08.pdf.
- [32] LunchBadger. API Gateway Comparison Guide, 10 2018. URL <https://www.lunchbadger.com/api-gateway-comparison-kong-enterprise-pricing-vs-express-gateway/>. Acedido a 2020-05-16.
- [33] LunchBadger. API Gateway: Express Gateway vs Tyk, 10 2018. URL <https://www.lunchbadger.com/vs-tyk-api-gateway-express-pricing/>. Acedido a 2020-05-16.
- [34] Mahesh Mahadevan. My experiences with API gateways..., 4 2019. URL <https://medium.com/@mahesh.mahadevan/my-experiences-with-api-gateways-8a93ad17c4c4>. Acedido a 2020-05-14.
- [35] Octávio José Azevedo Maia. CLAV: Autenticação e integração na plataforma iAP. Master's thesis, Universidade do Minho, 12 2019.
- [36] Rolando Santamaria Maso. A faster Node.js API Gateway for the masses (update 02/06/2019), 6 2019. URL <https://medium.com/sharenowtech/>

- [k-fastify-gateway-a-node-js-api-gateway-that-you-control-e7388c229b21](#).
Acedido a 2020-05-11.
- [37] Tim McLean. Critical vulnerabilities in JSON Web Token libraries, 3 2015. URL <https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/>. Acedido a 2019-12-22.
- [38] Faisal Memon. NGINX Controller API Management Module vs. Kong: Performance Comparison, 5 2019. URL <https://www.nginx.com/blog/nginx-controller-api-management-module-vs-kong-performance-comparison/>. Acedido a 2020-05-14.
- [39] MoleculerJS. API Gateway, 4 2020. URL <https://moleculer.services/docs/0.14/moleculer-web.html>. Acedido a 2020-05-15.
- [40] Natalya F. Noy and Deborah L. McGuinness. *Ontology Development 101: A Guide to Creating Your First Ontology*, 2000. URL https://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html. Acedido a 2020-05-01.
- [41] Ontotext. Architecture & Components, 12 2019. URL <http://graphdb.ontotext.com/documentation/free/architecture-components.html>. Acedido a 2020-01-09.
- [42] Ontotext. About GraphDB, 1 2020. URL <http://graphdb.ontotext.com/documentation/free/about-graphdb.html>. Acedido a 2020-01-09.
- [43] Passport.js. Overview, 2019. URL <http://www.passportjs.org/docs/>. Acedido a 2019-12-17.
- [44] Maria Celeste Pereira and Alexandra Lourenço. Apresentação de novo processo de negócio para integração na Lista Consolidada, 12 2016. URL http://arquivos.dglab.gov.pt/wp-content/uploads/sites/16/2017/08/FT3_Novo-PN.pdf. Acedido a 2020-06-01.
- [45] Sebastián E. Peyrott. *The JWT Handbook*. Auth0 Inc, 0.14.1 edition, 2018.
- [46] Ryan Pinkham. What Is the Difference Between Swagger and OpenAPI?, 10 2017. URL <https://swagger.io/blog/api-strategy/difference-between-swagger-and-openapi/>. Acedido a 2019-12-27.
- [47] Kristopher Sandoval. Top Specification Formats for REST APIs, 3 2016. URL <https://nordicapis.com/top-specification-formats-for-rest-apis/>. Acedido a 2019-12-31.

- [48] Marcus Schiesser. What is an API Gateway?, 1 2019. URL <https://glasnostic.com/blog/what-is-an-api-gateway-aws-express-kong>. Acedido a 2020-05-15.
- [49] The HTTPS-Only Standard. HTTP Strict Transport Security, 2020. URL <https://https.cio.gov/hsts/>. Acedido a 2020-06-12.
- [50] Swagger. What is Swagger?, 2019. URL <https://swagger.io/tools/open-source/getting-started/>. Acedido a 2019-12-27.
- [51] CheatSheets Series Team. HTTP Strict Transport Security Cheat Sheet, 2020. URL https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html. Acedido a 2020-06-12.
- [52] Mika Tuupola. Branca as an Alternative to JWT?, 8 2017. URL <https://appelsiini.net/2017/branca-alternative-to-jwt/>. Acedido a 2019-12-22.
- [53] Victoria University. Classification, 12 2014. URL https://library.vicu.utoronto.ca/archives/records_management/recordkeeping_manual/recordkeeping_101/classification_how_organize_your. Acedido a 2020-06-03.
- [54] Melroy van den Berg. Let's Encrypt – Is your website secure?, 12 2016. URL <https://blog.melroy.org/2016/lets-encrypt/>. Acedido a 2020-06-12.
- [55] Remy van Elst. Strong SSL Security on nginx, 06 2015. URL https://raymii.org/s/tutorials/Strong_SSL_Security_On_nginx.html. Acedido a 2020-06-12.
- [56] Ivan Vasiljevic. Adding Swagger To Existing Node.js Project, 8 2017. URL <https://blog.cloudboost.io/adding-swagger-to-existing-node-js-project-92a6624b855b>. Acedido a 2019-12-28.
- [57] Clara Viegas and Alexandra Lourenço. O que é a Lista Consolidada, 12 2016. URL http://arquivos.dglab.gov.pt/wp-content/uploads/sites/16/2017/08/FT2_LC.pdf. Acedido a 2020-06-01.
- [58] Clara Viegas and Alexandra Lourenço. Tabelas de seleção: aplicação no tempo, 2019. URL http://arquivos.dglab.gov.pt/wp-content/uploads/sites/16/2019/08/FT10_TS_Aplica%C3%A7%C3%A3o-no-tempo_2019-07-25.pdf. Acedido a 2020-06-01.

Este trabalho foi suportado pela Universidade do Minho (UM), projeto CLAV: Classificação e Avaliação da Documentação na Administração Pública Portuguesa (UMINHO/-BI/380/2019), financiado pela Direção-Geral do Livro, dos Arquivos e das Bibliotecas (DGLAB).