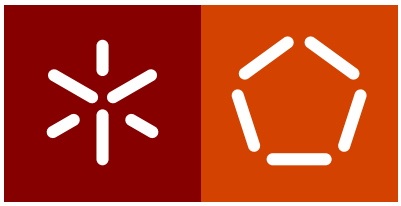


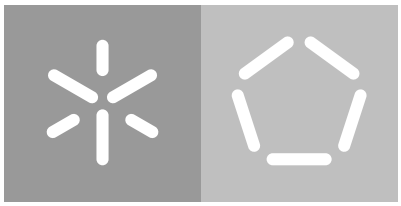
Universidade do Minho
Escola de Engenharia
Departamento de Informática

José Carlos Lima Martins

CLAV:
API de dados e Autenticação

Relatório de Pré-Dissertação

December 2019



Universidade do Minho
Escola de Engenharia
Departamento de Informática

José Carlos Lima Martins

CLAV:
API de dados e Autenticação

Relatório de Pré-Dissertação

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
José Carlos Leite Ramalho

December 2019

ABSTRACT

Write abstract here (en)

RESUMO

Escrever aqui resumo (pt)

CONTEÚDO

1 INTRODUÇÃO 3
1.1 Objetivos 3
2 ESTADO DA ARTE 5
2.1 Estado da Arte do CLAV 5
2.1.1 Estrutura 5
2.1.2 Formas de autenticação 6
2.2 JSON Web Token (JWT) 9
2.2.1 Estrutura do JWT 10
2.2.2 Criação de JWT/JWS 13
2.2.3 Alternativas ao JWT 14
2.3 Autorização de pedidos à API 16
2.3.1 Verificação dos *tokens* no servidor API 19
2.4 Autenticação.gov 21
2.5 MongoDB 21
2.6 Web Semântica 21
2.6.1 RDF 21
2.6.2 SPARQL 21
2.7 GraphDB 21
2.8 Swagger 21
2.9 Swagger-UI 21
3 O PROBLEMA E OS SEUS DESAFIOS 22
4 CONCLUSÃO 23

LISTA DE FIGURAS

Figura 1	Estrutura do CLAV incluíndo a interação de um utilizador com a mesma	6
Figura 2	Fluxo do *login* de um utilizador através do Autenticação.gov	9
Figura 3	Exemplo de representação compacta de JWT (quebra de linhas por forma a melhorar leitura)	10
Figura 4	Criação de um JWT	13
Figura 5	Criação de um JWS	14
Figura 6	Fluxo de autenticação e posteriores pedidos de um utilizador	17
Figura 7	Fluxo de autenticação e posteriores pedidos de uma chave API	18

LISTA DE TABELAS

LISTA DE EXEMPLOS

2.1 *Header* usado para construir o JWT da figura 3 11
2.2 *Payload* usado para construir o JWT da figura 3 12
2.3 *Signature* usado para construir o JWT da figura 3 12
2.4 Verificação se um pedido com uma determinada Chave API pode ser efetuado 19
2.5 Verificação se um pedido com um determinado *token* de um utilizador registado pode ser efetuado 19
2.6 Extração do *token* da *query string* 20
2.7 Extração do *token* da *heaer Authorization* 20
2.8 Verificação se um utilizador registado tem permissões suficientes para aceder a uma determinada rota 20

GLOSSÁRIO

Application Programming Interface Interface ou protocolo de comunicação entre um cliente e um servidor i

ontologia Representação de conhecimento (conceitos e as relações entre estes) 3

Simplex Programa de Simplificação Administrativa e Legislativa 3

LISTA DE ACRÓNIMOS

AEAD Authenticated Encryption with Additional Data 15
AP Administração Pública 3
API Application Programming Interface iii, iv, vi, 3–9, 15–21, *Glossary:* Application Programming Interface

CC Cartão de Cidadão 6, 8
CLAV Classificação e Avaliação da Informação Pública iii, iv, 3–8, 10, 15, 21
CSS Cascading Style Sheets 5
CSV Comma Separated Values 4

DGLAB Direção-Geral do Livro, dos Arquivos e das Bibliotecas 3, 7, 8

HMAC Hash-based Message Authentication Code 10, 11, 14, 15
HTML Hypertext Markup Language 5
HTTP Hypertext Transfer Protocol 16

IETF Internet Engineering Task Force 15

JOSE JSON Object Signing and Encryption 10, 11
JSON JavaScript Object Notation i, iii, 4, 9–15
JWE JSON Web Encryption 9, 10, 15
JWS JSON Web Signature iii, iv, 9, 10, 12–15
JWT JSON Web Token iii, iv, vi, 8–16

LC Lista Consolidada 3, 4

NIC Número de Identificação Civil 7
NSA National Security Agency 11

PDF Portable Document Format 5
POSIX Portable Operating System Interface 12

RDF Resource Description Framework 4
REST Representational State Transfer 3
RSA Rivest–Shamir–Adleman 10, 11

SAML Security Assertion Markup Language 14, 15
SHA-2 Secure Hash Algorithm 2 11
SWT Simple Web Token 14

UM Universidade do Minho 3

XML Extensible Markup Language 4, 14, 15

1

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. 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. Lourenço et al. (2019)

De forma a alcançar estes objetivos a Administração Pública (AP) tem desmaterializado processos e tem promovido a adoção de sistemas de gestão documental eletrónica bem como da digitalização de documentos destinados a serem arquivados. Lourenço et al. (2019)

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. Lourenço et al. (2019)

Nasce assim o projeto Classificação e Avaliação da Informação Pública (CLAV) com 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 Simplex. Lourenço et al. (2019)

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. Lourenço et al. (2019)

1.1 OBJETIVOS

A continuação do desenvolvimento da API de dados da CLAV nesta dissertação, seguindo uma metodologia REST, 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 extramemente essencial que a API de dados do CLAV possua uma boa documentação ajudando futuros programadores ou utilizadores a utilizar a API. 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 AMA (2019) 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 no 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 um API Gateway simplifica a comunicação entre interface/utilizadores e a API de dados.

Resumidamente, os objetivos desta dissertação são:

- Documentação em *Swagger* da API de dados da CLAV
- 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
- Proteção da API de dados da CLAV com múltiplos níveis de acesso
- Estudo da crição de um API Gateway
- Integração do CLAV no iAP

2

ESTADO DA ARTE

2.1 ESTADO DA ARTE DO CLAV

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 do CLAV quando esta dissertação iniciou aprofundando principalmente os pontos mais importantes sobre o tema desta dissertação.

2.1.1 *Estrutura*

O CLAV está dividido em duas partes:

- interface (*front-end*) presente em `http://clav.dglab.gov.pt`
- API de dados (*back-end* que inclui também duas bases de dados, *GraphDB* e *MongoDB*) presente em `http://clav-api.dglab.gov.pt`.

Cada parte encontra-se numa máquina diferente.

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 do 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 e os pedidos à API serão feitos do computador do utilizador para o servidor da API de dados e não do servidor da interface para o servidor da API de dados. Ou seja, o fluxo de cada um desses pedidos será igual ao fluxo no caso em que se acede diretamente a API sem uso de qualquer interface.

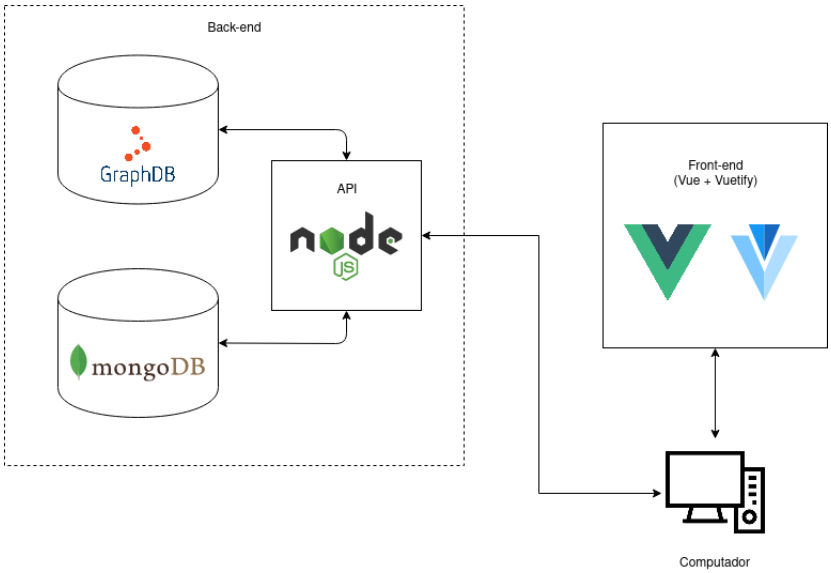


Figura 1: Estrutura do CLAV incluíndo a interação de um utilizador com a mesma

2.1.2 *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 ou 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 autenticarem-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 do CLAV.

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

- Nível 0: Chaves API
- Nível 1: Representante Entidade
- Nível 2: Utilizador Simples
- Nível 3: Utilizador Avançado
- Nível 3.5: Utilizador Validador (AD)

- Nível 4: Utilizador Validador
- Nível 5: Utilizador Decisor
- Nível 6: Administrador de Perfil Funcional
- Nível 7: Administrador de Perfil Tecnológico

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 terá de ter esta hierarquia em conta.

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 nesta a partilha dessa informação com a plataforma do CLAV. A *password* é armazenada não na sua forma literal mas sim a sua *hash* ao aplicar 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`[1].

Durante esta tese com a proteção da API 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 por forma a utilizadores representantes de outras entidades se registarem na plataforma terão de: DGLAB (2019)

- Preencher o formulário disponibilizado para o efeito, para cada representante designado pela entidade;

1 Para mais informação veja *rainbow table attack*

- 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.

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. 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. Passport.js (2019) Tem como única preocupação a autenticação delegando qualquer outra funcionalidade para a aplicação que a usa. Este *middleware* possui muitas estratégias de autenticação entre as quais a local (`email/username + password`), JWT, *OAuth*[2], *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 do CLAV). O fluxo do *login* neste caso é:

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

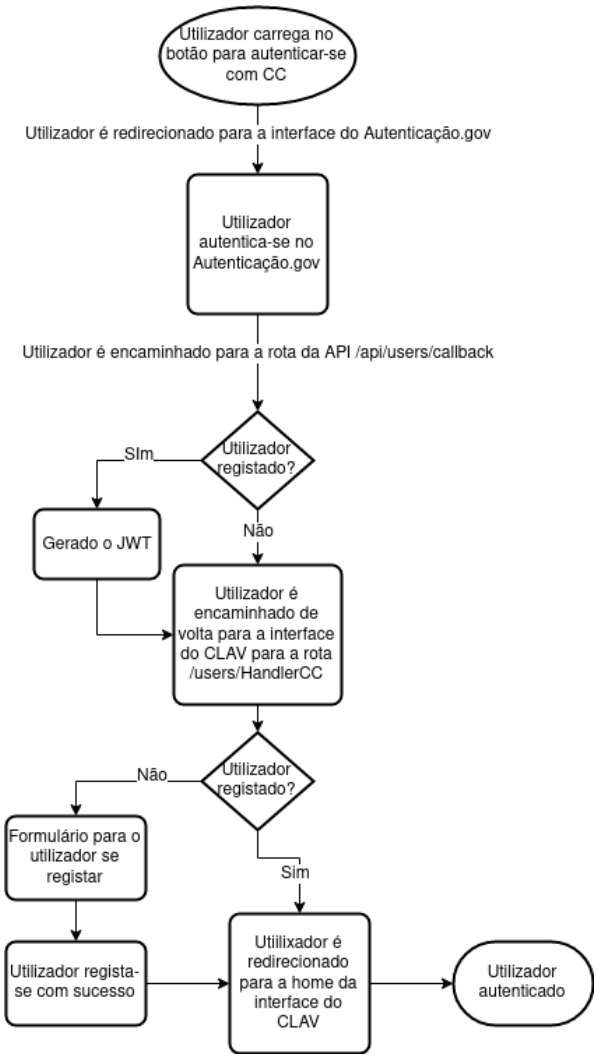


Figura 2: Fluxo do *login* de um utilizador através do Autenticação.gov

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

2.2 JSON Web Token (JWT)

O JWT é um *open standard*[3] que define uma forma compacta e independente de transmitir com segurança informação entre partes com um objeto JSON. Autho (2019) O JWT pode ser assinado digitalmente (JWS), encriptado (JWE), assinado e depois encriptado (JWS encriptado, ou seja, um JWE, ordem recomendada[4]) ou encriptado e depois assinado (JWE assinado, ou seja, um JWS).

3 Mais informação em https://tools.ietf.org/html/rfc7519
4 Mais informação em https://tools.ietf.org/html/rfc7519#section-11.2

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 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.

2.2.1 *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`[5]). As codificações `Base64 URL-safe` de cada elemento são depois concatenadas através de pontos dando origem a uma representação final compacta do JWT (*JWS/JWE Compact Serialization*). Na secção 2.2.2 está presente dois diagramas referentes à construção de dois JWTs sendo um deles assinado.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJuYW1lIjoiSm9zw6kgTWFydGlucyIsIm51bSI6ImE3ODgyMSJ9.
tRPSYVsFI-nziRPuAjdGZLN2tUez5MtLML_aAnPplgM

Figura 3: 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 3) consiste nos seguintes atributos:

5 Variante da codificação `Base64` onde a codificação gerada é segura para ser usada em *URL*s. Basicamente para a codificação `Base64` gerada substitui os caracteres '+' e '/' pelos caracteres '-' e '_' respetivamente. Além disso, remove o caracter de *padding* e proibe separadores de linha

- O atributo obrigatório (único campo obrigatório para o caso de um JWT não encriptado) `alg` (algorítmo) onde é indicado que algoritmo é usado para assinar e/ou desencriptar. O seu valor pode ser por exemplo HS256 (HMAC com o auxilio do SHA-256[6]) 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* conter atributos arbitrários este atributo não deve ser colocado. Caso o *payload* for um JWT[7] 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 2.1: *Header* usado para construir o JWT da figura 3

***Payload*:** O *payload* (a roxo na figura 3) 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*): Peyrott (2018)

- `iss` (*issuer*): Identificador único (*case-sensitive string*) que identifica unicamente quem emitiu o JWT. A sua interpretação é específica a cada aplicação visto que não há uma autoridade central que gere 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.

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

7 JWT aninhado (*nested JWT*)

- `exp` (*expiration* (*time*)): Um número inteiro que representa uma data e hora específica no formato *seconds since epoch* definido pela POSIX[8], 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 específica no mesmo formato do atributo `exp`, mas que a partir da qual o JWT é considerado válido.
- `iat` (*issued at* (*time*)): Um número inteiro que representa uma data e hora especifica no mesmo formato dos atributos `exp` e `nbf` na 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 2.2 (`name` e `num`) são atributos privados.

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

Exemplo 2.2: *Payload* usado para construir o JWT da figura 3

***Signature*:** A assinatura (a azul da figura 3) 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
)
```

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

Exemplo 2.3: *Signature* usado para construir o JWT da figura 3

2.2.2 *Criação de JWT/JWS*

Na figura 4 é apresentada a construção de um JWT em que o atributo `alg` (algorítmo) tem o seu valor igual a `none`, ou seja, o JWT não é assinado nem encriptado.

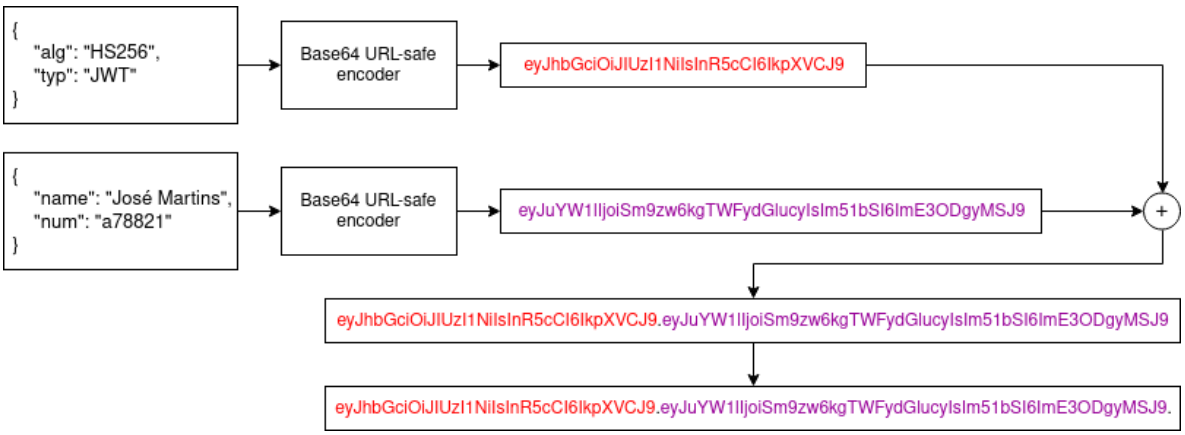


Figura 4: Criação de um JWT

Já na figura 5 é demonstrada a construção de um JWT assinado, ou seja, um JWS.

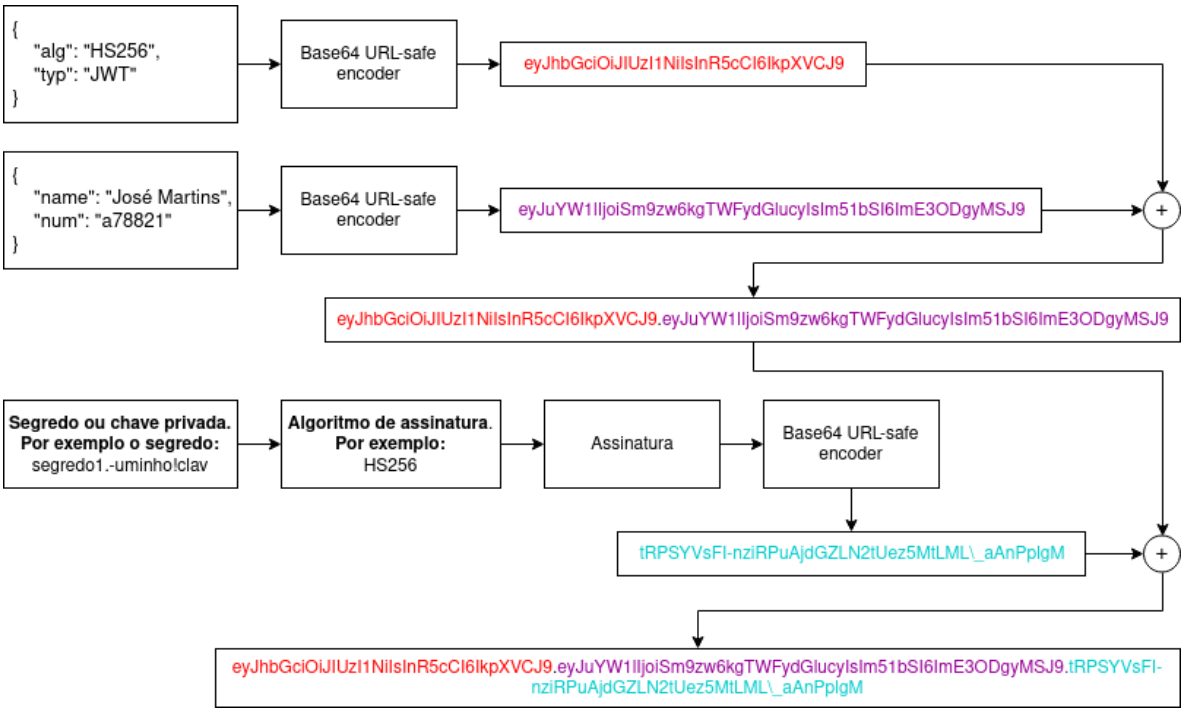


Figura 5: Criação de um JWS

2.2.3 *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 dificil quando comparado com a simplicidade de assinar JSON. Autho (2019)

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 com um algoritmo e com recurso a um segredo/chave privada.

- As bibliotecas ao fazer a verificação seja 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 recebe 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 McLean (2015) bem como deve ser *deprecated* e deixar de ser incluído nos JWTs[9].

A biblioteca que será usada na CLAV, `jsonwebtoken`, já endereçou estes problemas[10] pelo que estas vulnerabilidades não estarão presentes na CLAV.

Por outro os *parsers* de JSON são mais comuns em grande parte das linguagens de programação visto que os JSONs mapeiam diretamentepara objetos ao contrário do XML que não tem um mapeamento natural de documento para objeto. Autho (2019) Portanto isto torna mais fácil trabalhar com JWT do que com SAML.

Já quando comparamos os JWT 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. Mais à frente na secção **??** veremos outra possível alternativa com recurso a API *gateways* em que uma possível abordagem é dentro da API *gateway* ser através de JWTs, fora ser usado *sessionIDs* e na "entrada" da API *gateway* ter uma base de dados que associa os *sessionsIDs* aos JWTs.

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 e que garantam integridade. Tem também uma região de *payload* como JWT com a única diferença é que este *payload* não tem um 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. Tuupola (2017) 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

9 Ver https://gist.github.com/paragonie-scott/c88290347c2589b0cd38d8bb6ac27c03
10 Ver https://github.com/auth0/node-jsonwebtoken/commit/1bb584bc382295eeb7ee8c4452a673a77a68b687

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.

2.3 Autorização de pedidos à API

Quanto à forma como os pedidos serão feitos à API poderão ser feitos de duas formas, através da *header* HTTP *Authorization* ou através da *query string* do pedido em um 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* irá se usar o esquema de autenticação *Bearer*[11] com umas pequenas alterações. Portanto o conteúdo da *header Authorization*:

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

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

Convém referir que a Chave API é também um *token*. A divisão entre utilizadores e chaves API permite uma mais fácil gestão dos *tokens* recebidos pela API bem como usar duas formas diferentes de os gerar/verificar com o possível benefício de melhorar a segurança da API.

Os *token*s gerados pela API serão JWTs. Contudo poderiam ser outro tipo de *tokens* (por exemplo uma *string* aleatória e única) que o processo de envio dos *tokens* para a API manter-se-ia igual.

Após descrito como poderão ser feitos os pedidos à API, irá ser apresentado 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 será o seguinte:

11 Mais informação em `https://tools.ietf.org/html/rfc6750`

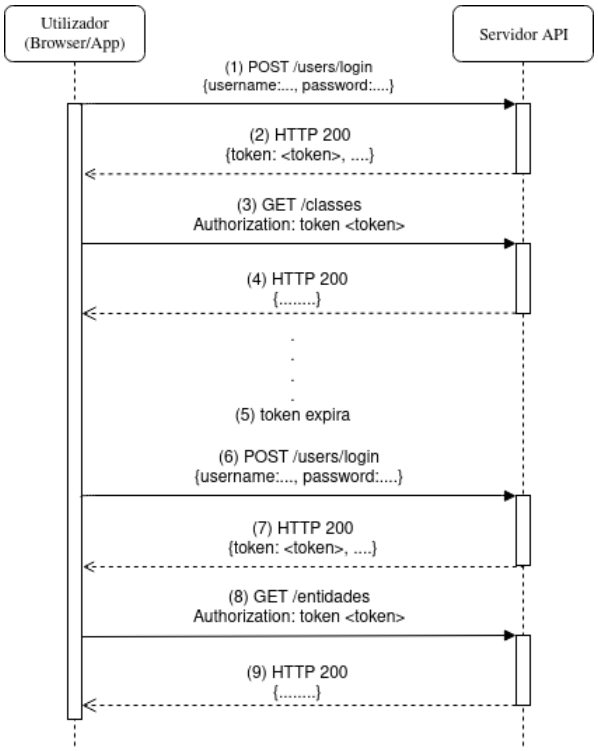


Figura 6: Fluxo de autenticação e posteriores pedidos de um utilizador

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 será o seguinte:

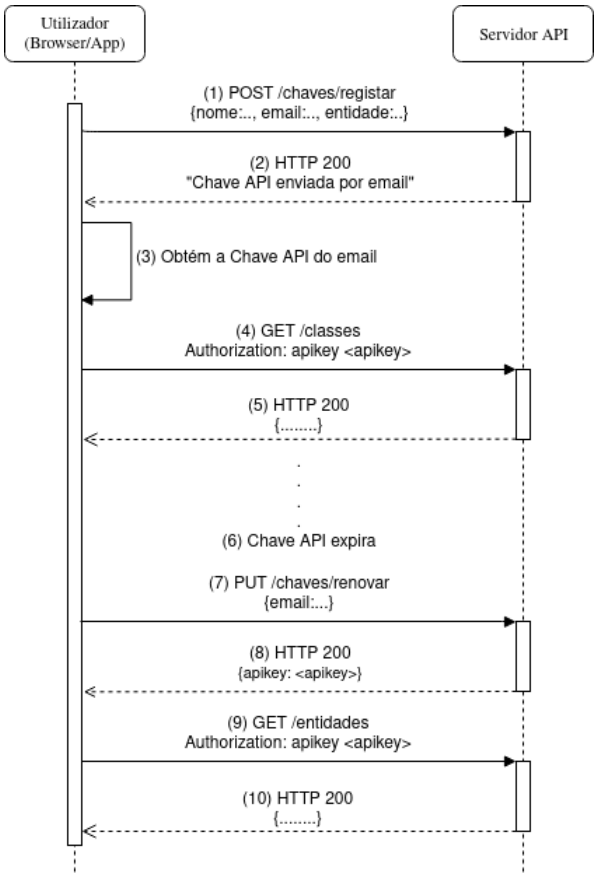


Figura 7: Fluxo de autenticação e posteriores pedidos de uma chave API

1. Utilizador cria uma chave API ao providenciar o nome, email e entidade
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 do 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

2.3.1 *Verificação dos tokens no servidor API*

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. De seguida será apresentado o pseudo-código de verificação dos *tokens* tendo em conta que os utilizadores registados conseguem aceder a todas as rotas que as Chaves API conseguem mas que o inverso não acontece. Ou seja, um utilizador registado até o de nível mais baixo por exemplo, consegue aceder a todas as rotas que as Chaves API tem acesso e mais algumas nas quais as Chaves API não têm permissões de acesso.

Por forma a validar se uma Chave API pode aceder a uma determinada rota pode ser executada a seguinte função em *middleware*:

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

    if key then
        keyBD = getKeyFromMongoDB(key)
        if keyBD then
            res = jwt.verify(key, secretForAPIkey)
            if res != expired then
                if keyBD.active == True then
                    return next()
                else
                    return err
            else
                return err
        else
            return err
    else
        return isLoggedInUser(req, res, next)
```

Exemplo 2.4: 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 registado pode aceder a uma determinada rota pode ser executada a seguinte função em *middleware*:

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

    if key then
        res = jwt.verify(key, secretForToken)
        if res != expired then
```

```
            if keyBD.active == True then
                return next()
            else
                return err
        else
            return err
    else
        return err
```

Exemplo 2.5: Verificação se um pedido com um determinado *token* de um utilizador registado pode ser efetuado

A obtenção do *token* bem como a verificação deste *token* pode ser obtido através da utilização da estratégia do `passport` chamada `passport-jwt`.

Além disso as obtenções dos *tokens* tanto das Chaves API como de *tokens* de utilizadores registados podem ser obtidos através da utilização de extratores presentes na estratégia `passport-jwt`. 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 2.6: Extração do *token* da *query string*

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

```
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 2.7: Extração do *token* da *heaer 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 também em *middleware* a seguinte função:

```
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
```

```
            return err
```

Exemplo 2.8: 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 ao superior ao `clearance`.

Com estas três funções (`isLoggedInKey`, `isLoggedInUser` e `checkLevel`) 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.

2.4 AUTENTICAÇÃO.GOV

AMA (2018)

2.5 MONGODB

Satheesh et al. (2015)

2.6 WEB SEMÂNTICA

DuCharme (2011)

2.6.1 *RDF*

DuCharme (2011)

2.6.2 *SPARQL*

DuCharme (2011)

2.7 GRAPHDB

2.8 SWAGGER

2.9 SWAGGER-UI

3

O PROBLEMA E OS SEUS DESAFIOS

4

CONCLUSÃO

BIBLIOGRAFIA

AMA. *Autenticação.gov - Fornecedor de autenticação da Administração Pública Portuguesa*, 1.5.1 edition, 12 2018.

AMA. Autenticação.gov, 2019. URL https://autenticacao.gov.pt/fa/Default.aspx. Acedido a 2019-11-20.

Autho. Introduction to json web tokens, 2019. URL https://jwt.io/introduction/. Acedido a 2019-12-19.

DGLAB. CLAV - Classificação e Avaliação da Informação Pública, 2019. URL http://clav.dglab.gov.pt. Acedido a 2019-12-15.

Bob DuCharme. *Learning SPARQL*. O'Reilly, 1st edition, 7 2011. ISBN 978-1-449-30659-5.

Alexandra Lourenço, José Carlos Ramalho, Maria Rita Gago, and Pedro Penteado. Plataforma CLAV: contributo para a disponibilização de dados abertos da Administração Pública em Portugal. Acedido a 2019-11-20, 7 2019. URL http://hdl.handle.net/10760/38643.

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.

Passport.js. Overview, 2019. URL http://www.passportjs.org/docs/. Acedido a 2019-12-17.

Sebastián E. Peyrott. *The JWT Handbook*. 0.14.1 edition, 2018.

Mithun Satheesh, Mithun Satheesh, and Jason Krol. *Web Development with MongoDB and NodeJS*. Packt Publishing, 2nd edition, 10 2015. ISBN 978-1-78528-752-7.

Mika Tuupola. Branca as an alternative to jwt?, 8 2017. URL https://appelsiini.net/2017/branca-alternative-to-jwt/. Acedido a 2019-12-22.

NB: place here information about funding, FCT project, etc in which the work is framed. Leave empty otherwise.