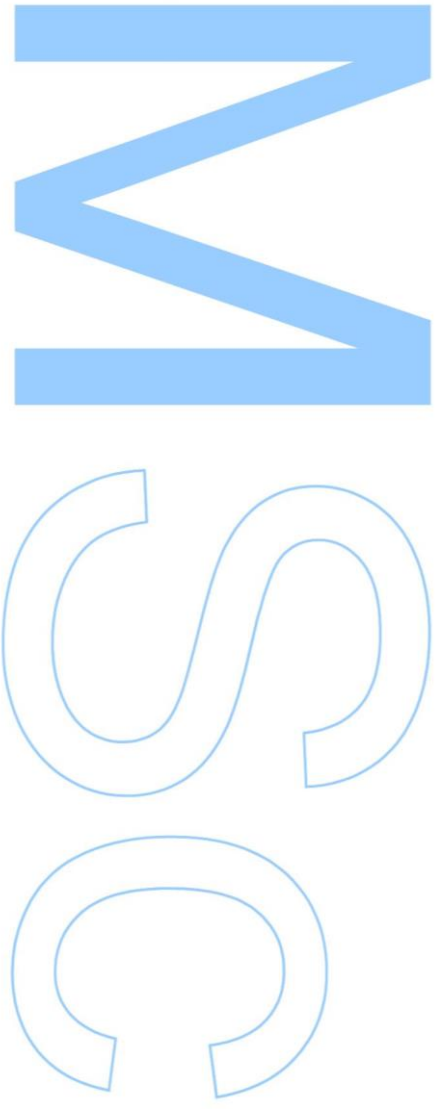
U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Portal de Validação de Documentos Assinados

David de Paula Santos Silva
Mestrado em Segurança Informática
Faculdade de Ciências da Universidade do Porto
2021

Orientador
Rogério Reis, FCUP

Supervisor
Nuno Santos, Loqr



U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Abstract

The commercial, governmental, and religious contract acceptance throughout the human history was made of drawings, seals, and stamps concepts. A wax seal stamped on a letter ensured the document was authentic, so only a family member would have that stamp. According to history, blood usually was used, and for instance, The Sandugo, a friendship agreement between the Spanish explorer Miguel López de Legazpi and Datu Sikatuna, chieftain of Bohol in the Philippines on March 16, 1565. Over the years and the written improvement, these acceptance concepts were replaced by the well-known signature. For a while the authenticity, integrity validation of a document with signature base was a lengthy bureaucratic process. Nevertheless, the development of new technologies and different cryptographic methods could be designed the digital signature system, a way that helps clients and companies to sign documents digitally, saving response time, reducing paper consumption in the office and safeguarding the documents authenticity and integrity. Due to this constant increase in the use of digital signatures and the limited offer of portals that present the result of validation in a simple and intuitive way for less technical users, the proposal for this project arose. The aim of this project is the development of a system composed by the user interface, front-end, that will allow to upload a file in digitally signed PDF, and through the validations in back-end, validate if the current signature is valid, sending back a brief of the signature status and the full report. Besides the aim is to simplify language in which the outcome is presented to the user that will accept or not, as it demands experience to a regular computer user to understand it.

Keywords: Digital Signature; Documents; Validation; Authenticity; Integrity.

Resumo

No decorrer da história humana, a aceitação de acordos comerciais, governamentais, religiosos eram feitos através dos conceitos de desenhos, selos e carimbos. O selo de cera estampado em uma carta garantia que o documento era autêntico, pois somente alguém da família teria acesso à aquele carimbo selador. Como é conhecido na História, algumas vezes até mesmo sangue era utilizado, como é o caso do Sandugo, um acordo de amizade entre o navegador espanhol Miguel López de Legazpi e Datu Sikatuna, chefe na ilha de Bohol nas Filipinas em 1565. Com o passar dos anos e da aprimoração da escrita, estes conceitos de aceitação foram substituídos pelo que hoje conhecemos por assinatura. Durante algum tempo, a validação da autenticidade e integridade de um documento com base na assinatura foi um processo demorado e burocrático. Entretanto, com o desenvolvimento de novas tecnologias e diferentes métodos criptográficos, foi possível criar o sistema de assinatura digital, uma solução que ajuda clientes e empresas à assinarem documentos digitalmente, a economizar tempo de resposta, a reduzir volume de papel em escritórios e a salvaguardar a autenticidade e integridade dos documentos. Devido à este constante aumento de utilização de assinaturas digitais e da pouca oferta de portais que apresentam o resultado da validação de forma simples e intuitiva para utilizadores menos técnicos, surgiu a proposta deste projeto. Este trabalho tem como objetivo o desenvolvimento de um sistema composto pela interface com o utilizador, *front-end*, que permitirá realizar a submissão de um ficheiro em formato PDF assinado digitalmente, e através de validações ocorridas no *back-end*, validar se a assinatura presente no documento é válida, devolvendo o resumo do estado da assinatura e o relatório detalhado. Além disto, têm-se como objetivo simplificar a linguagem na qual o resultado é apresentado para o utilizador para decidir a aceitação do mesmo, visto que é demasiado técnica para o entendimento de um utilizador de informática comum.

Palavras chaves: Assinatura Digital; Documentos; Validação; Autenticidade; Integridade.

Agradecimentos

Obrigado a todos que contribuíram para o desenvolvimento desta tese, tanto de forma direta, como indireta. Aos meus pais (in memoriam) por nunca me fazerem desistir dos meus sonhos e por sempre me incentivaram a estudar. Aos meus irmãos, Emanuele Bastos e Reginaldo de Paula, por darem suporte quando foi necessário. Aos meus sobrinhos, Pablo e Hadassa, por trazerem alegria em momentos de angústia e fraqueza. Ao meu melhor amigo, Matheus Reis, por oferecer todo apoio durante esta jornada. Aos meus *housemates*, Goga Gavric, Filipe Ribeiro e Ricardo Melo, por serem minha família em terras Portuguesas. Aos meus amigos, Rodrigo Ananias, Paula Araújo, Guilherme Reis, Taissa Venancio, Larissa Fernandes, Laion Nascimento, Antonio Amaral, Diego Souza, Roberto Araújo, Caio Farias, Clarissa Vannuchi, Ygor Soares, Silas Damasceno, Mariana Miranda e Igor Wanick por compreenderem e apoiarem a minha ausência em alguns momentos durante o desenvolvimento deste trabalho. Ao meu orientador Professor Rogério Reis, pela paciência e por me auxiliar nos momentos de dificuldade. Ao Professor Manuel Barbosa, pela dedicação e cuidado ao lecionar. E ao Nuno Santos e Helder Souza da Loqr, por contribuirem nas questões tecnológicas e na resolução de dúvidas. Vocês todos foram fundamentais nesta jornada. Muito obrigado.

Dedico esta tese aos meus amados pais (in memoriam).

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Blocos de Código

Acrónimos

AES Advanced Electronic Signature

API Application Programming Interface

AS Trust Services

CA Certification Authority

CAdES CMS Advanced Electronic Signatures

CMS Cryptographic Message Syntax

CRLs Certificate Revocation Lists

CSS Cascading Style Sheets

CSV Comma-separated values

DAC Data Authentication Code

DSS Document Security Store

eID Electronic Identification

eIDAS Electronic Identification, Authentication and Trust Services

ERDS Electronic Registered Delivery Service

ES Electronic Signature

ETSI European Telecommunications Standards Institute

FCUP Faculdade de Ciências da Universidade do Porto

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

ISO International Organization for Standardization

JPA Java Persistence API

JSP JavaServer Pages

JWT JSON Web Tokens

LDAP Lightweight Directory Access Protocol

LTV Long Term Validation

MACs Message Authentication Codes

MVC Model-View-Controller

Oauth Open Authorization

OWASP Open Web Application Security Project

PAdES PDF Advanced Electronic Signatures

PAdES-BES PAdES Basic Electronic Signature

PAdES-EPES PAdES Explicit Policy Electronic Signature

PAdES-LTV PAdES Long Term Validation

PDF Portable Document Format

PKI Public Key Infrastructure

QES Qualified Electronic Signature

QSCD Qualified Electronic Signature Creation Device

QWAC Qualified Web Authentication Certificate

RA Registration Authority

REST Representational State Transfer

TSP Trust Service Provider

URL Uniform Resource Locator

VRI Validation Related Information

XAdES XML Advanced Electronic Signatures

XML Extensible Markup Language

Capítulo 1

Introdução

1.1 Apresentação do Projeto

Este projeto é resultado da parceria entre a Faculdade de Ciências da Universidade do Porto (FCUP) e a empresa Loqr, e tem como objetivo a criação de um portal de validação de documentos assinados, de forma a aliar o ambiente de pesquisa da universidade ao cenário do mercado de trabalho da Loqr.

A proposta do projeto, de modo geral, foi definida como o desenvolvimento de um portal que efetue a validação de assinaturas digitais e que apresente de forma simplificada o resultado de maneira que qualquer utilizador com conhecimentos limitados desta tecnologia possa aferir a sua validade e decidir a aceitação das mesmas. De um modo mais específico, seria o desenvolvimento de um portal, incluindo *front-end* e *back-end*, que permita a um utilizador submeter um Documento de Formato Portátil - Portable Document Format (PDF) assinado digitalmente, obtendo como resultado a visualização do documento, uma representação visual simplificada da validação da assinatura, e por fim um relatório básico e um relatório mais detalhado da validação da assinatura.

O portal resultante deste projeto permite aumentar consideravelmente a usabilidade de documentos assinados digitalmente em formato PDF pela sua abrangência a qualquer dispositivo com navegador, aumentando também a abrangência em termos público alvo.

1.2 Metodologia

A metodologia utilizada neste projeto consistiu em seis fases, a saber: a **primeira fase** tratou do *kick-off* do projeto entre as partes envolvidas, no qual foram apresentadas as principais necessidades e os eventuais desafios. A **segunda fase** tratou da pesquisa bibliográfica e estado da arte, de forma a perceber o funcionamento de um sistema de validação de assinaturas digitais e dos atuais *softwares* para isto. A **terceira fase** foi a elaboração dos requisitos funcionais e não funcionais, o desenho das *features* do sistema em Gherkin, além da elaboração do diagrama de

fluxo de dados. A **quarta fase** do projeto é dividida em três partes: a primeira parte consistiu na criação da interface *front-end* para o utilizador submeter o ficheiro, a segunda consistiu no desenvolvimento do *back-end* para tratar das validações da assinatura do ficheiro PDF de forma simplificada e a terceira parte tratou da integração entre o *front-end* e *back-end* com Spring Boot. A **quinta fase** corresponde ao período de testes da solução, ajustes de código e análise de segurança. E por fim, a **sexta fase** abordou a análise dos resultados e respectivas conclusões.

1.3 Organização

Esta seção apresenta a disposição dos capítulos do projeto seguido pelo conteúdo de cada um, respetivamente.

No Capítulo 2 são apresentados os conceitos básicos para o entendimento do conteúdo deste trabalho, como funções *hash*, criptografia, assinatura digital, certificados digitais, infraestrutura de chave pública, desenvolvimento *web*, Transferência Representacional de Estado - Representational State Transfer (REST) e Interface de Programação de Aplicações - Application Programming Interface (API).

No Capítulo 3 é apresentado o estado da arte, no qual são citadas as limitações entre diferentes programas e *websites* que permitem a validação de assinaturas digitais e a solução a ser desenvolvida, além de apresentar quais serão os principais objetivos finais.

No Capítulo 4 serão abordados os passos necessários para o desenvolvimento da solução para o portal de validação de assinaturas digitais, assim como a parte de interação com o utilizador (*front-end*). Na primeira parte será exposta a análise de requisitos funcionais, não funcionais e de segurança, em seguida, o diagrama alto nível, o diagrama de blocos do processo de validação do *DSS Bundle*, a arquitetura Modelo-Vista-Controlador - Model-View-Controller (MVC), o desenvolvimento do *front-end*, e por fim, o desenvolvimento do *back-end*.

No Capítulo 5 serão apresentados as experiências e testes com vários exemplos de ficheiros com diferentes quantidades e tipos de assinaturas, a preparação do ambiente de execução, o processo de utilização da aplicação, o *download* do relatório básico e avançado e a análise de segurança com o *software* Sonarqube.

No Capítulo 6 serão apresentados os resultados e análises dos diferentes ficheiros enviados para testes, também será abordada a performance da aplicação e do *DSS Bundle* para realizarem as validações, os erros encontrados e a quantidade de vulnerabilidades que foram identificadas através do Sonarqube.

Por fim, no Capítulo 7 serão apresentadas as conclusões obtidas através da realização deste projeto, nas quais são divididas em alguns pontos: resumo da solução, objetivos concluídos, desafios encontrados, trabalhos futuros e considerações finais.

Capítulo 2

Conceitos Básicos

2.1 Criptografia

Historicamente, a criptografia se divide em: Clássica e Moderna. O período clássico compreende desde os povos antigos até a invenção de máquinas eletrónicas. Acredita-se que a 4000 anos atrás, um escriba do Egito Antigo aplicou o "conceito" de criptografia dentro da tumba de Khnumhotep II, na qual foram feitos símbolos hieroglíficos menos comuns de forma a transmitir dignidade e autoridade para o nobre faraó. A inscrição não era secreta, entretanto incorporou um dos elementos essenciais da criptografia: a transformação deliberada da escrita, e é conhecido como o texto mais antigo conhecido por fazê-lo. Foi durante o período moderno, em especial o da Segunda Guerra Mundial, que a criptografia realmente começa a se desenvolver, considerando que era extremamente necessário proteger as informações sobre bases e alvos, e, para isto, foi utilizada a máquina conhecida como Enigma que permitia substituir cada caracter por outro, através de um sistema de rotores que serviam de chave para cifrar a mensagem, conforme cita David Khan [19].

A criptologia é a disciplina científica que estuda os conhecimentos matemáticos, computacionais, etc. reunindo técnicas da criptografia e da criptoanálise. A criptoanálise é o ramo da criptologia que estuda formas de decifrar uma mensagem sem conhecer a chave secreta [25]. O estudo da forma para cifrar e decifrar dados é chamada criptografia. O fundamento principal da criptografia é permitir que duas partes, normalmente chamadas de Alice e Bob, troquem informações (texto, dados numéricos ou absolutamente nada — a estrutura é completamente arbitrária), num canal inseguro, no qual um observador não consiga perceber o que está a ser dito. A força criptográfica de um sistema é medida através do tempo e dos recursos necessários para se recuperar o *plaintext*. Para realizar o procedimento de cifrar e decifrar é utilizado um algoritmo criptográfico, ou também chamado de *cipher*. Segundo Network Associates [18], a segurança do dado cifrado depende de dois elementos, a robustez do algoritmo criptográfico e do espaço de quantidade de chaves utilizadas para cifrar.

2.1.1 Criptografia Simétrica

Na criptografia simétrica, a mesma chave é utilizada tanto para cifrar como para decifrar, esta chave é acordada previamente através de um canal seguro, e justamente esta distribuição de chave é um dos problemas deste sistema, como realizar a entrega da chave para o destinatário sem que ninguém a intercepte. O funcionamento básico da criptografia simétria de acordo com Kostas Zoto et.al. [31] acontece da seguinte forma: Alice cifra a informação, também chamada de *plaintext*, à ser transmitida, utilizando a chave pré-determinada, e envia o texto cifrado (*ciphertext*) pelo canal seguro. Caso Eve esteja a espiar o canal, não conseguirá perceber a messagem inicial, entretanto, Bob como possui a chave, consegue decifrar e reconstruir o texto original, conforme é ilustrado na Figura 2.1.

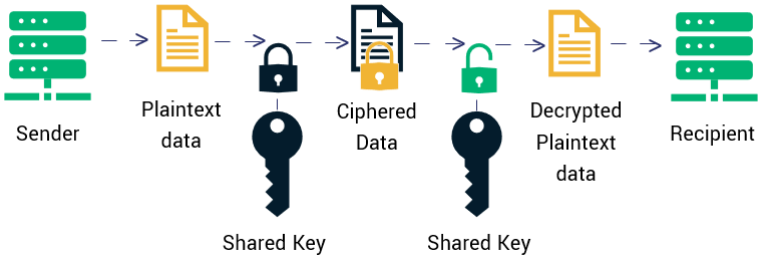


Figura 2.1: Esquema de funcionamento da criptografia simétrica.[1]

2.1.2 Criptografia de Chave Pública - Public Key Infrastructure (PKI)

Já na criptografia de chave pública, o problema da distribuição de chaves é resolvido, para cada utilizador do sistema existe um par de chaves: uma chave privada que deve ser mantida em segredo e que é utilizada para decifrar, e uma chave pública que pode ser disponibilizada, numa repositório de chaves por exemplo, e utilizada para cifrar os dados, ou seja, somente a pessoa com a chave privada correspondente à chave pública consegue decifrar os dados [18], ilustrado na Figura 2.2.

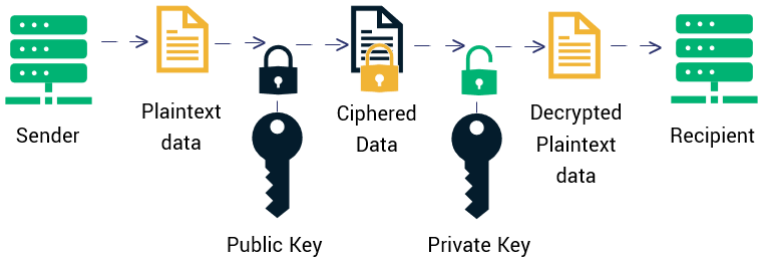


Figura 2.2: Esquema de funcionamento da criptografia de chave pública.[2.1]

[1] Fonte: https://sectigostore.com/blog/types-of-encryption-what-to-know-about-symmetric-vs-asymmetric-encryption/

Apesar de resolver o problema da troca de chaves, a criptografia de chave pública apresenta velocidade reduzida quando aplicada em grandes quantidades de informações, entretanto a criptografia de chave pública é considerada mais segura.

2.1.2.1 Infraestrutura de Chave Pública

A principal característica de uma PKI, Figura 2.3 abaixo, é a introdução da Autoridade Certificadora - Certification Authority (CA) e dos componentes da Autoridade de Registo - Registration Authority (RA) [18].

Uma infraestrutura de chave pública é um conjunto de *hardware*, *softwares*, processos, e procedimentos nos quais são utilizados para gerenciar, criar, alterar, revogar, utilizar e distribuir certificados digitais e chave pública.

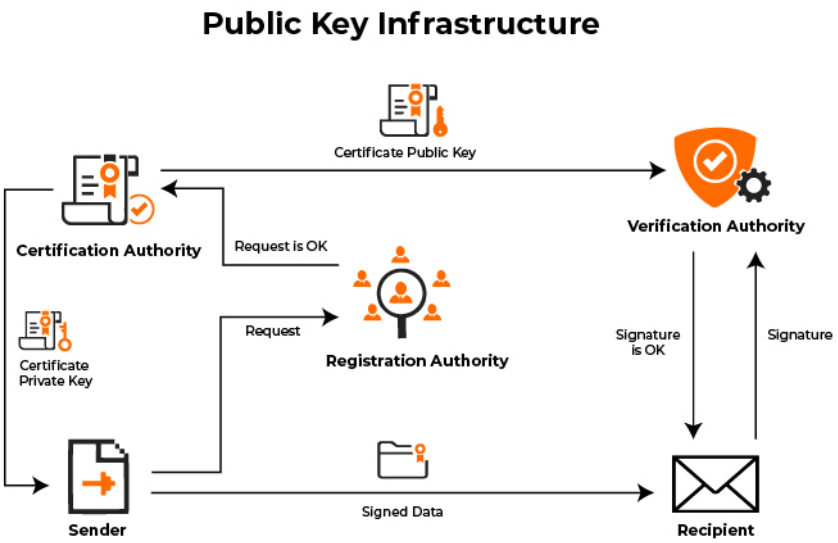


Figura 2.3: Esquema de funcionamento de uma PKI.[2]

A CA é um terceiro confiável responsável por estabelecer uma cadeia hierárquica de confiança, manter e emitir Listas de Certificados Revogáveis - Certificate Revocation Lists (CRLs), além de criar certificados X.509 e assiná-los digitalmente utilizando a chave privada, de forma a certificar a identidade dos utilizadores. Por causa disto, é o elemento central de uma PKI. Ao utilizar a chave pública da CA, é possível verificar a autenticidade do certificado, além da integridade do conteúdo.

A CA numa PKI também é chamada de raiz de confiança e possui um certificado autoassinado, visto que ela está no topo da hierarquia. Através desta estrutura, todos os certificados gerados pelas CA subordinadas assinados pela CA raiz, são ditos confiáveis, conforme é dito por Reshma [1].

[2]Fonte: https://www.appviewx.com/education-center/pki/

A Figura 2.4 apresenta a hierarquia de certificados composta por uma CA raiz.

Figura 2.4: Hierarquia de certificados composta por uma CA raiz.[3]

Por outro lado, de acordo com Luxtrust [21], a RA é entidade humana responsável por apoiar o registo de utilizadores com a PKI, e também controlar se os dados pessoais que vão aparecer no certificado são os mesmos documento de identificação. Essa verificação garante a emissão de um certificado que realmente representa a identidade digital confiável do titular. Adicionalmente, a RA também gerencia o ciclo de vida do certificado no caso de um titular legítimo solicitar uma revogação.

Para além disto, existe também o chamado *Cross Certification*, que permite que entidades de uma PKI confiem noutras entidades de outra PKI. De acordo com SANS [17], é um método de interconexão entre duas PKI, de forma a construir uma cadeia de certificação numa grande comunidade. Essa relação de confiança mútua é definida por um acordo de certificação cruzada entre a CA de cada PKI, entretanto, apesar da geração de *cross-certificates* ser fácil de ser realizada, os processos e técnicas essencias para garantir a confiança entre as duas partes é complexo. A Figura 2.5 ilustra o esquema de funcionamento do *Cross Certification*.

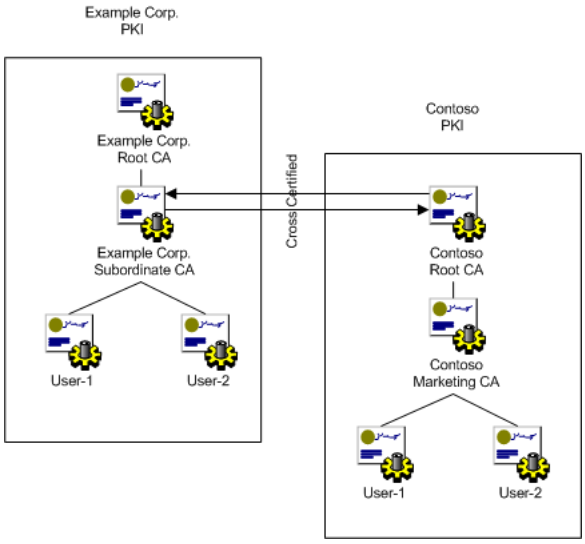


Figura 2.5: Esquema de funcionamento do cross certification.[4]

[3]Fonte: https://docs.microsoft.com/en-us/windows/win32/seccertenroll/about-certificate-hierarchy

Formalmente, para compreender a infraestrutura de chave pública é necessário perceber a definição de um esquema de criptografia. Na qual, de acordo com Steven Galbraith [12], é definido por:

Seja $\kappa \in \mathbb{N}$ um parâmetro de segurança, um esquema de criptografia é definido pelos seguintes espaços abaixo, todos dependentes do parâmetro de segurança κ e dos algoritmos.

- M_k: o espaço de todas mensagens possíveis;
- PK_k: o espaço de todas chaves públicas possíveis;
- SK_k: o espaço de todas chaves privadas possíveis;
- C_k: o espaço de todos textos cifrados possíveis;
- KeyGen: um algoritmo que recebe o parâmetro de segurança κ, executa num tempo polinomial, como, O(K^c) operações de *bit* para alguma constante c $\in \mathbb{N}$. E, como resultado, tem-se uma chave pública p$\kappa \in PK_k$, e uma chave privada s$\kappa \in SK_k$;
- Cifrar: um algoritmo randômico que recebe como entrada m $\in M_k$ e pK, executa num tempo polinomial, e como resultado tem-se um texto cifrado c $\in C_k$;
- Decifrar: um algoritmo que normalmente não é randômico, que recebe como entrada c $\in C_k$ e sK, executa num tempo polinomial, e como resultado tem-se um texto decifrado m $\in M_k$ ou então, em caso de erro, o valor apresentado pelo símbolo $\perp$;

É necessário que: Decifrar(Cifrar(m,pk),sk) = m, se (pk,sk) são um par de chaves.

2.1.2.2 Esquema de Identificação de Chave Pública

A criptografia de chave pública também pode ser utilizada para assinaturas digitais. O processo no qual um Verificador confirma a identidade que um Comprovador alega ser, é chamado de identificação ou autenticação de entidade, conforme menciona Gennaro et.al. [8]. Informalmente, o protocolo entre o Comprovador e o Verificador é chamado de esquema de identificação de chave pública, na qual o Confirmador possui uma chave pública pk e uma chave privada sk, e o Verificador possui apenas a cópia de pk. Basicamente, o protocolo é dividido em 3 passos [12]:

- Passo 1: O Comprovador envia um "comprometimento" para o Verificador, chamado S_0;
- Passo 2: O Verificador envia um desafio S_1;
- Passo 3: O Comprovador envia S_2;

[4]Fonte: https://docs.microsoft.com/en-us/windows/win32/seccertenroll/about-cross-certification

Desta forma, o Verificador aceita ou rejeita a prova enviada. O protocolo serve para convencer o *Verifier* que ele realmente está a se comunicar com alguém que conhece a chave privada correspondente à chave pública do Comprovador. Diferente da criptografia de chave pública, com os esquemas de identificação e de assinaturas digitais, o utilizador está sempre a produzir resultados computacionais envolvendo a sua chave privada. Deste modo, é necessário garantir que nenhuma informação relativa à chave privada seja exposta. Por fim, ainda de acordo com Gennaro [8], o protocolo de identificação tem dois objetivos principais:

- *Complitude*: em caso de entidades honestas, o Comprovador consegue se autenticar para o Verificador com sucesso.
- *Robutez*: em caso de um Comprovador desonesto, a probabilidade de convencer um Verificador é praticamente nula.

2.1.3 Funções *Hash* e Códigos Autenticadores de Mensagens - Message Authentication Codes (MACs)

As funções *hash* tem um papel muito importante dentro da criptografia, particularmente na criptografia de chave pública, pois são utilizadas em funções de derivação de chaves, assinaturas digitais e MACs.

Uma função de *hash* ou *digest* é uma função na qual um conjunto de dados, independente do tamanho, é transformado numa série de caracteres de comprimento fixo. A função *hash* muito provavelmente se houver alguma modificação numa único *bit*, o resultado da operação será totalmente diferente. Os dados que serão transformados em *hash* recebem o nome de mensagem ou pré-imagem, e o conjunto de todas possíveis mensagens ou pré-imagens é denominado Domínio de Mensagens ou Espaço de Mensagens [5]. De um modo geral, o modo de funcionamento de uma função *hash*, ilustrado na Figura 2.6, é descrito por:

- A mensagem inicial (*input*) é dividido em blocos.
- O *hash* do primeiro bloco é calculada e obtêm-se um valor fixo.
- O *hash* do segundo bloco é calculada e combinada com o *hash* obtido anteriormente.
- O processo é repetido e finalizado até que todos os blocos sejam calculados.

Também é dito por Bruce Schneier [27], que as funções *hash* são relativamente fáceis de calcular, mas significativamente mais difíceis de reverter, ou seja, dado x, é fácil de calcular f(x), entretanto dado f(x), é difícil calcular x. Neste caso, o conceito de difícil se baseia no tempo computacional necessário para que se consiga chegar ao resultado de x, dado f(x).

[5]Fonte: https://tracer.lcc.uma.es/problems/avalanche/avalanche.html

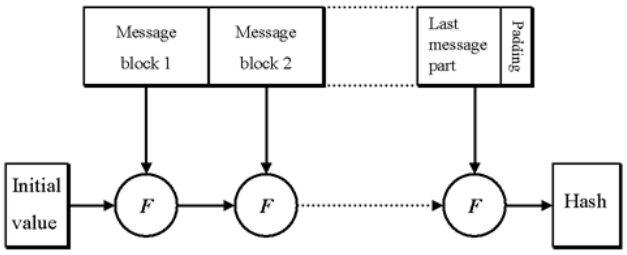


Figura 2.6: Esquema de funcionamento Merkle-Damgaard para uma função hash.[5]

Formalmente, e de acordo com Steven Galbraith [12], uma função *hash* é uma função que mapeia objetos, palavras de comprimento finito arbitrário para palavras de tamanho fixo l, ou seja, é uma operação criptográfica que transforma dados de tamanho variável numa valor de tamanho fixo. Já uma família de funções *hash*, é um conjunto de funções $\{H_k : k \in \kappa\}$, para um conjunto finito de κ, no qual cada função da família é escrita na forma de $H_k : \{0,1\}^* \Rightarrow \{0,1\}^*$. Além disto, o valor κ que é específico na função da família *hash* H_k é chamado de chave, e em algumas aplicações este valor não é mantido secreto.

Para além disto, o nível de segurança que as funções *hash* têm que apresentar são definidas através das seguintes propriedades:

- Resistência a Pré-imagem: Dado um *hash* com valor h, deve ser difícil encontrar qualquer mensagem m que seja h=*hash*(m). Este conceito é relacionado com as funções *one-way*, ou seja, é fácil de calcular o *output*, entretanto é difícil reverter a imagem de um *input* randômico.
- Resistência a Segunda Pré-imagem: Dado uma palavra x e uma palavra $y = H(x)$, deve ser computacionalmente inviável encontrar $x' \neq x$, de modo que $H(x') = y$.
- Resistência a Colisões: Deve ser computacionalmente inviável encontrar uma palavra $x \neq x'$, de modo que $H(x) = H(x')$. Ou seja, deve ser extremamente difícil encontrar quaisquer duas mensagens que produzam o mesmo resultado.

Os MACs, conhecidos também como Código de Autenticação de Dados - Data Authentication Code (DAC), são funções que incorporam uma chave secreta e são utilizados para autenticar mensagens através de uma *tag* incorporada, assim como garantir a integridade. A função *hash* resultante é uma combinação da pré-imagem e da chave secreta [27]. Se o remetente não souber a chave secreta, o valor do *hash* será diferente, o que diria ao destinatário que a mensagem não foi enviada pelo remetente original. A diferença crucial entre MACs e assinaturas digitais é que os MACs usam uma chave simétrica k para gerar a etiqueta de autenticação e verificá-la [6]. Assim como as funções *hash*, os MACs também apresentam propriedadades importantes [30]:

- Compressão: h_k mapeia o *input* x de tamanho finito de *bit* arbitrário para um *output* h(x) de tamanho de *bit* fixo n, entretanto aplica H_k, onde k que é a chave secreta.

- Fácil de Calcular: Como o nome da propriedade indica, os MACs também são fáceis de calcular, ou seja, dado h e um *input* x, h(x) é fácil de calcular.
- Resistência ao Cálculo: Sem conhecer a chave secreta, é computacionalmente impossível descobrir que $H_k(x)$ corresponde à um x, mesmo com amostras dos resultados do algoritmo.

Matematicamente, de acordo com Christof Paar e Jan Pelzl [6], os MACs são uma função de chave simétrica k e mensagem x, dada por:

$$m = MAC_k(x)$$

O princípio de verificação e cálculo dos MACs é destacado na Figura 2.7. Conforme se pode observar, Bob calcula o Código de Autenticação de Mensagem em função da mensagem m e da chave secreta x partilhada, em seguida, envia para Alice a mensagem m com a *tag* de autenticação incorporada. Ao receber a mensagem, Alice realiza o processo de verificação de ambos. Visto que é um processo simétrico, basta realizar os mesmos passos de Bob, ela recalcula a *tag* de autenticação com a mensagem recebida e a chave secreta partilhada.

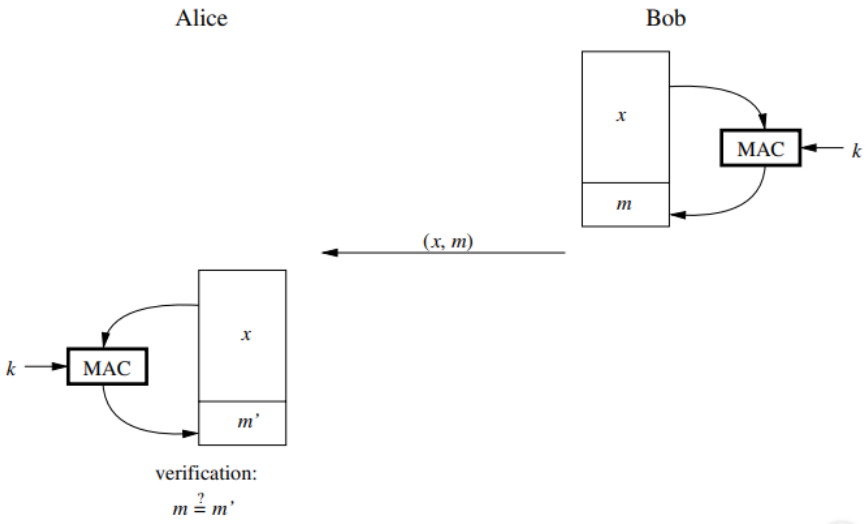


Figura 2.7: Princípio de verificação e cálculo dos MACs.[6]

Desta forma, Alice tem certeza que Bob enviou a mensagem, considerando que somente ambos tem a chave partilhada para calcular os MACs. Caso um atacante tivesse intercetado a mensagem e modificado o conteúdo, ele não conseguiria calcular, pois lhe faltaria conhecer a chave secreta.

2.2 Certificados Digitais

Os certificados digitais são credenciais que facilitam a verificação de identidades entre os utilizadores numa transação, ou seja, uma identidade virtual com garantias de autenticidade e

[6]Fonte: https://link.springer.com/chapter/10.1007%2F978-3-642-04101-3_12

proteção das informações enviadas, sendo o X.509 o certificado mais utilizado na Internet. X.509 é um formato padrão para certificados de chave pública, documentos digitais que associam com segurança pares de chaves criptográficas a identidades como sites, indivíduos ou organizações. O certificado X.509 também é utilizado em vários protocolos, como o TLS/SSL, que é a base do Protocolo de Transferência de Hipertexto Seguro - Hypertext Transfer Protocol Secure (HTTPS). De acordo com a definição dada por Cátia Gomes [15], os certificados digitais consistem num documento assinado de forma criptográfica por uma autoridade de certificação ou auto-assinado, de forma a associar uma determinada chave pública à uma entidade (pessoa, organização, aplicação, entre outros). De forma geral, a sua principal função é gerar confiança numa sistema que faz a utilização da estrutura de chave pública, pois certifica as chaves das entidades.

2.2.1 Estrutura de um Certificado Digital

Um certificado digital possui diversas informações importantes em sua estrutura, tais como:

- A versão do certificado;
- A chave pública da entidade especificada no certificado;
- Informações da entidade (nome, email, etc.);
- A data de validade do certificado;
- A assinatura da entidade/CA que garante que a chave pública ali contida realmente é fidedigna às informações que estão presentes;
- O emissor do certificado;
- O nome do algoritmo criptográfico de assinatura utilizado pela CA.

Desde sua criação em 1998, já existiram 3 versões dos certificados digitais, entretanto a cada versão lançada, os campos anteriores da estrutura permanecem e os novos campos são adicionados, conforme ilustrado na Figura 2.8.

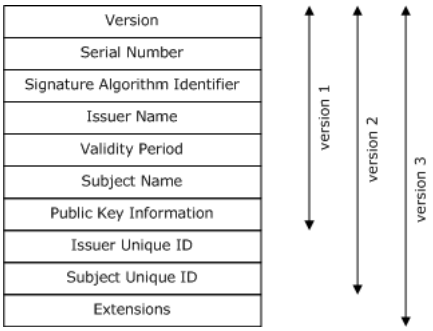


Figura 2.8: Estrutura básica de um certificado digital de acordo com as suas versões.[7]

[7]Fonte: https://docs.microsoft.com/en-us/windows/win32/seccertenroll/about-x-509-public-key-certificates

Já a Figura 2.9 representa um certificado digital da empresa Google.

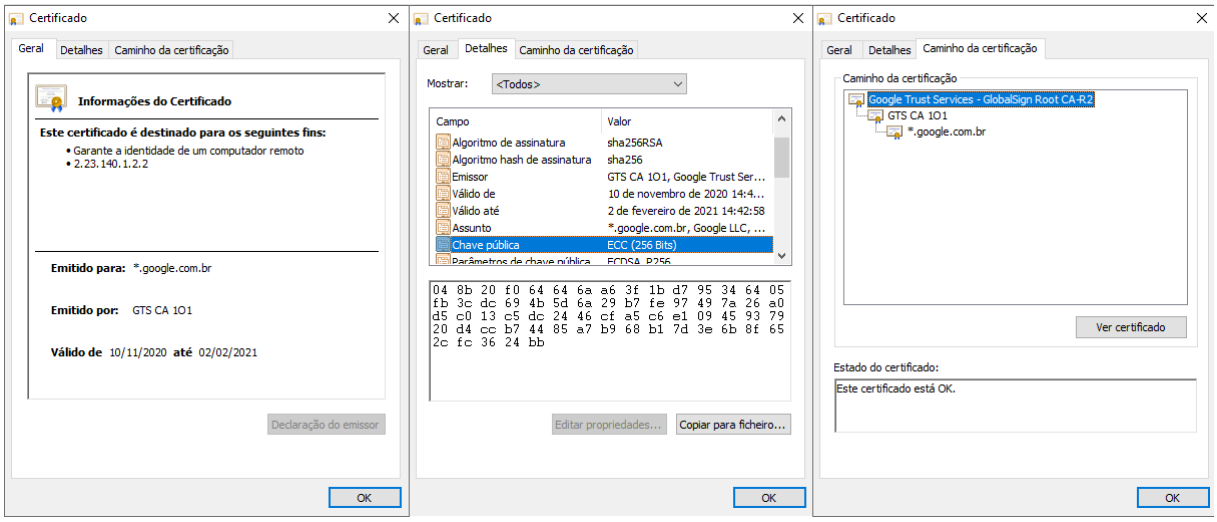


Figura 2.9: Exemplo de um certificado digital.

2.2.2 Tipos de Certificados

Basicamente, os Certificados Digitais possuem duas funções, assegurar que as pessoas, *sites*, etc. são fontes confiáveis e também fornecer proteção para as informações trocadas. Segundo Reshma [1], existem 5 tipos de certificados digitais de acordo com a utilização:

1. **Classe 1 - Pessoais**: Utilizados para segurança de *e-mails* pessoais;
2. **Classe 2 - Organizacional**: Utilizados internamente dentro de empresas para identificar empregados e segurança de *e-mails*;
3. **Classe 3 - Servidores**: Utilizados pelos donos de *sites* para garantir aos seus utilizadores que os mesmos estão a aceder o local correto e de sua propriedade;
4. **Classe 4 - Transações Online**: Utilizados para transações *online* entre companhias;
5. **Classe 5 - Governamentais**: Utilizados para segurança governamental.

Para certificados com utilização de maiores riscos, como por exemplo o de transações eletrónicas, o nível de validação pela *CA* é bem maior quando comparado aos certificados de utilização em *e-mails*, a quantidade de validações é relativamente baixa [1].

2.2.3 Processo de Obtenção de um Certificado Digital pela CA

O processo de obtenção de um certificado digital pode ser realizado solicitando para uma CA comercial. Deste modo, a CA será responsável por gerar o par de chaves em nome do utilizador

e incorporar a chave pública ao certificado, seguido por todas as validações necessárias definidas na política da CA. A chave privada deve ser mantida em segredo, e caso seja comprometida, o utilizador deve notificar a CA para que a mesma seja revogada. É através dos CRLs que as CA conseguem mapear a lista de certificados cujas chaves privadas tenham sido revogados ou perdidas. A Figura 2.10 abaixo destaca o processo mencionado acima.

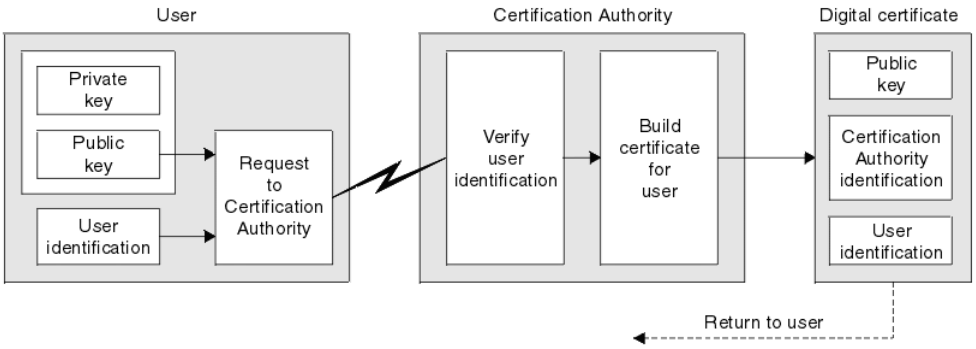


Figura 2.10: Processo de obtenção de um certificado digital.[8]

2.2.4 Processo de Validação de um Certificado Digital

O processo de validação de um certificado digital *online* é executado pelos navegadores para garantir que os certificados sejam válidos e tenham um caminho de certificação válido, conforme é descrito por Reshma [1], se dá através dos seguintes passos:

1. Quando um utilizador tenta aceder um *website*, o certificado digital é enviado do servidor para o navegador;
2. Ao receber o pedido do servidor, o navegador verifica se a CA que assinou o certificado é confiável, visto que possui uma lista de CA confiáveis (*stores* de certificados) instalado e suas respetivas chaves públicas;
3. O navegador utiliza a chave pública do certificado para decifrar a assinatura no certificado da empresa e obtém um *hash*;
4. Um *hash* dos conteúdos do certificado também é realizado;
5. Se ambos *hashes* forem iguais, então a assinatura do certificado é validada para ser assinada pela CA confiável e a chave pública presente é dita válida;
6. O navegador também verifica se o certificado ainda está válido, ou seja, não esteja expirado;
7. Por fim, o nome do certificado é validado novamente com o nome do *website*. Se forem iguais, então uma conexão segura é estabelecida para a realização de transações *online*.

O processo de validação de um certificado digital pode ser observado na Figura 2.11.

[8]Fonte: https://sites.google.com/site/ddmwsst/digital-certificates
[9]Fonte: http://cs.indstate.edu/~rafshar/documents/paper2.pdf

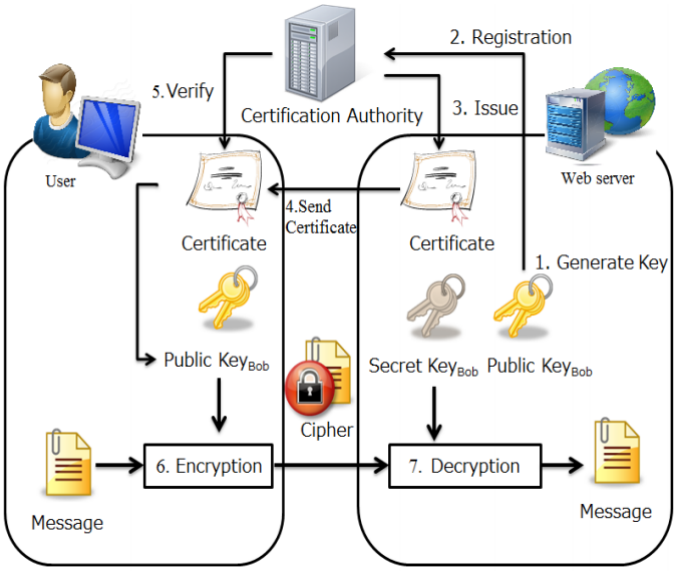


Figura 2.11: Processo de validação de um certificado digital online.[9]

Para além disto, quando um certificado é criado por uma CA comercial, existe a Cadeia de Validação ou também chamado de Caminho de Validação, que consiste literalmente numa hierarquia de validação, na qual o navegador realiza uma busca pela chave pública da CA emissora ou de uma CA intermediária que assinou o certificado, Figura 2.12. Este processo é realizado até que a CA raiz auto-assinada seja encontrada e que navegador acredite que seja confiável.

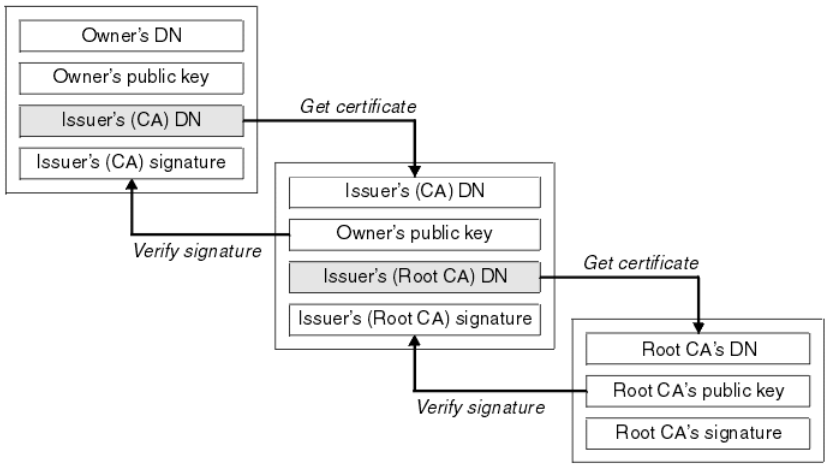


Figura 2.12: Cadeia de validação de um certificado digital online.[10]

Por fim, na cadeia de validação, cada certificado também possui um código baseado em sua implementação que permite validar que:

- Se assinatura é válida;
- Se as datas de início e fim estão configuradas corretamente;

[10]Fonte: https://sites.google.com/site/ddmwsst/digital-certificates

- Se certificado está expirado;
- Se o certificado foi revogado;
- Se algum dos certificados mais altos na hierarquia de PKI expirou;
- Se o certificado está a ser utilizado para uma finalidade diferente da pretendida.

Com base nesse código, cada certificado possui uma precedência atribuída à ela, na qual um certificado expirado tem precedência mais alta que um certificado revogado [22]. Desta forma, sabe-se que um certificado expirado não deve ser verificado quanto ao *status* de revogação, e o *status code* com a precedência mais alta será aplicado à cadeia de certificados e propagado para o *status* da cadeia de certificados. Na Figura 2.13 pode-se observar uma cadeia de hierarquia de certificados do *site* Overleaf.

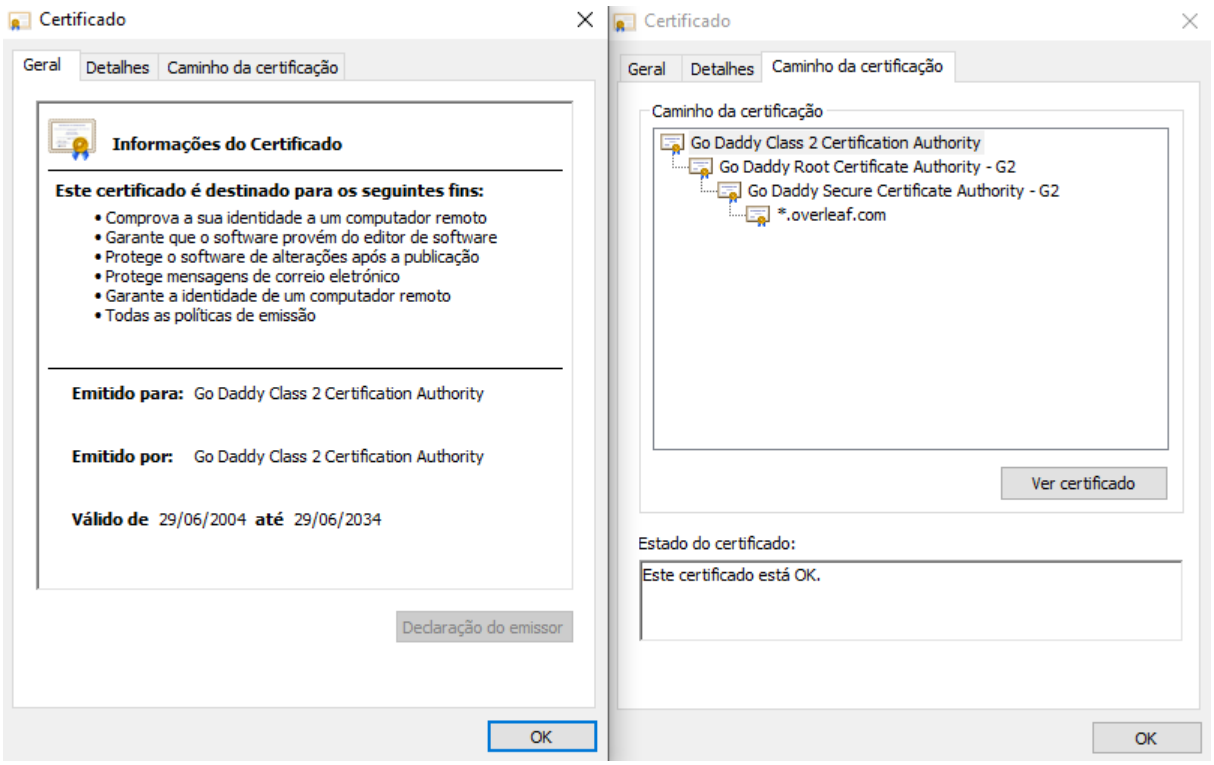


Figura 2.13: Exemplo de cadeia de hierarquia de certificados.

2.3 Assinaturas Digitais

Primeiramente é necessário compreender a diferença entre algoritmos determinísticos, aleatorizados e perfeitos. De acordo com Steven Galbraith [12], um algoritmo determinístico é aquele utilizado para resolver problemas computacionais se não fizer uso de nenhuma aleatoriedade, e que devem terminar depois de um número finito de passos, independentemente do seu *input*. Já um algoritmo aleatorizado, pode correr sem garantia de terminar, se a sequência das escolhas

aleatórias não for boa. E por fim, um algoritmo perfeito é aquele que sempre o resultado é correto.

Formalmente, um esquema de assinaturas é definido através de uma mensagem, assinatura e espaços de chaves dependentes do parâmetro κ, mencionado no tópico anterior e apresentado por Steven Galbraith [12] da seguinte forma:

- **Assinar**: um algoritmo aleatorizado que é executado em tempo polinomial, e recebe como *input* a mensagem m e a chave privada sk, e como resultado tem a assinatura s.
- **Verificar**: um algoritmo normalmente determinístico que é executado em tempo polinomial, e recebe como *input* a mensagem m, a assinatura s e a chave pública pk, e como resultado tem-se a assinatura classificada como válida ou inválida.

A função Verificar(m, Assinar(m, sk),pk) deve ser válida, com ínfima probabilidade de um falso positivo.

Um dos maiores benefícios da criptografia de chave pública é a possibilidade de se implementar assinaturas digitais. De acordo com Mehran Alidoost Nia et. al. [7], uma assinatura digital prova a identidade do seu proprietário de forma que o remetente não possa alegar que ele não enviou as informações, além de permitir verificar que não houve alteração durante a transmissão. Segundo Network Associates [18], a assinatura digital é melhor em relação a assinatura manual, pois é quase impossível de ser falsificada, além de atestar o conteúdo das informações, bem como a identidade do signatário.

As assinaturas digitais apresentam 3 principais funções para uma comunicação segura [15].

- **Integridade:** Através da aplicação de funções de *hash*, qualquer mudança no documento dará origem a um *hash* diferente.
- **Autenticidade:** Ao utilizar chave pública para verificação, é possível identificar o autor da assinatura;
- **Não Repúdio:** Ao se aplicar o método de chave pública para cifrar, o signatário não pode constestar que não tenha assinado, pois somente ele possui a chave privada.

2.3.1 Processo de Criação e Verificação de uma Assinatura Digital

O processo de criação de uma assinatura digital pressupõe na verificação da identidade e distribuição da chave privada para o utilizador signatário pelo Provedor Confiável de Serviço - Trust Service Provider (TSP) ou também chamado de Serviços de Confiança - Trust Services (AS), que são responsáveis por garantir a identificação eletrónica dos signatários e serviços, utilizando mecanismos fortes de autenticação, certificados digitais e assinaturas eletrónicas.

O TSP emite e assina um Certificado Digital, incluindo a chave pública do signatário e suas informações. É gerada um *hash* de tamanho pré-definido do documento chamado *Digest* que, posteriormente, é cifrada com a chave privada do signatário. Por fim, este *hash/digest* cifrado é combinado com o documento original, de forma a produzir o documento assinado digitalmente [24], Figura 2.14.

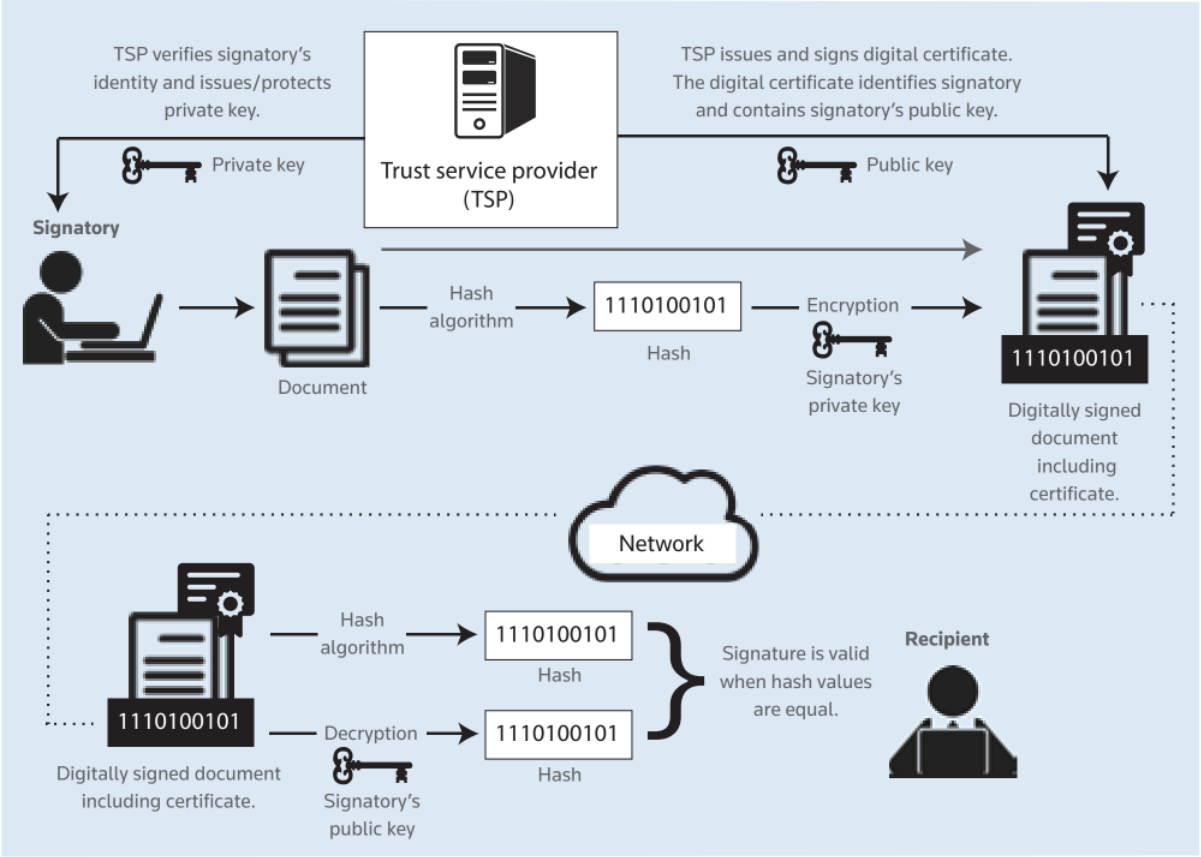


Figura 2.14: Processo de criação e verificação de uma assinatura digital.[11]

Para a verificação, o processo ocorre de modo similar. De um lado é gerado o *hash* do documento recebido, e do outro, utilizando a chave pública do signatório, é gerada outro *hash* com base na assinatura digital no documento. Ambos os *hashes* são comparados, e caso sejam iguais, a assinatura é valida e o documento é dito válido.

2.3.1.1 Indicações de Resultado do Processo de Verificação de Assinaturas Digitais

Uma aplicação que realiza a verificação de assinaturas digitais pode retornar diferentes resultados para a assinatura [9], tais como: Válida, Inválida ou Indeterminada, conforme se pode observar na Tabela 2.1, é importante ressaltar que estas são apenas recomendações.

[11] Fonte: https://ec.europa.eu/futurium/en/system/files/ged/plc_article_on_e-signature_platforms_final_feb_2017.pdf

[12] Fonte: Adaptado de https://www.etsi.org/deliver/etsi_ts/102800_102899/102853/01.01.02_60/ts_102853v010102p.pdf

Tabela 2.1: Indicação do resultado da assinatura e o seu significado [12]

Indicação do Resultado	**Significado**
VÁLIDA	A assinatura é tecnicamente válida com base nas seguintes considerações: • A assinatura é criptograficamente válida e • Quaisquer restrições aplicáveis à identidade do signatário a certificação foi validada positivamente (por exemplo, o certificado do signatário, foi considerado confiável), e • A assinatura foi validada positivamente contra as restrições de validação e, portanto, é considerada em conformidade com essas restrições.
INVÁLIDA	A assinatura é inválida com base na falha de pelo menos uma das considerações acima.
INDETERMINADA	A informação disponível é insuficiente para apurar se a assinatura é VÁLIDA ou INVÁLIDA.

A Tabela 2.2 destaca as sub-indicações caso o resultado do processo de validação da assinatura digital seja Inválido ou Indeterminado.

Tabela 2.2: Indicações dos sub-resultados de assinaturas inválidas ou indeterminadas e o seu significado.[13]

Indicação do Resultado	**Sub-indicação do Resultado**	**Significado**
INVÁLIDA	REVOKED	A assinatura é considerada inválida porque: 1) O certificado do signatário foi encontrado para ser revogado e 2) O algoritmo de validação da assinatura pôde verificar que a hora da assinatura está após o tempo de revogação.
INVÁLIDA	HASH_FAILURE	A assinatura é considerada inválida porque pelo menos um hash de um objeto de dados assinados incluído no processo de assinatura não corresponde ao valor de hash correspondente na assinatura.

Continua na próxima página

Tabela 2.2 – continuação da página anterior

Indicação do Resultado	Sub-indicação do Resultado	Significado
INVÁLIDA	SIG_CRYPTO_FAILURE	A assinatura é considerada inválida porque o valor da assinatura na assinatura não pôde ser verificada usando a chave pública do assinante no certificado do assinante.
INVÁLIDA	SIG_CONSTRAINTS _FAILURE	A assinatura é considerada inválida porque uma ou mais propriedades da assinatura não corresponde às restrições de validação.
INVÁLIDA	CHAIN_CONSTRAINTS _FAILURE	A assinatura é considerada inválida porque a cadeia de certificados usada no processo de validação não corresponde às restrições de validação relacionadas ao certificado.
INVÁLIDA	EXPIRED	A assinatura é considerada inválida porque o Algoritmo de Validação de Assinatura pode verificar se a hora de assinatura é posterior à data de expiração (não após) do certificado do assinante.

Continua na próxima página

Tabela 2.2 – continuação da página anterior

Indicação do Resultado	Sub-indicação do Resultado	Significado
INVÁLIDA	CRYPTO_CONSTRAINTS _FAILURE	A assinatura é considerada inválida porque pelo menos um dos algoritmos que foram usados em um material (por exemplo, o valor da assinatura, um certificado ...) envolvido na validação da assinatura ou o tamanho das chaves usadas com tal algoritmo não é mais considerado confiável e o Algoritmo de Validação de Assinatura pode verificar se esse material foi produzido após o tempo até o qual o algoritmo foi considerado seguro.
INVÁLIDA	NOT_YET_VALID	A assinatura é considerada inválida porque o Algoritmo de Validação de Assinatura pôde verificar se a hora de assinatura é anterior à data de emissão (não antes) do certificado do signatário.
INVÁLIDA	FORMAT_FAILURE	A assinatura encontrada não apresentou conformidade com base nos padrões de formato.
INVÁLIDA	POLICY_PROCESSING _ERROR	Um determinado ficheiro de política formal não pôde ser processado por qualquer motivo (por exemplo, não acessível, não analisável, etc.)

Continua na próxima página

Tabela 2.2 – continuação da página anterior

Indicação do Resultado	Sub-indicação do Resultado	Significado
INVÁLIDA	UNKNOWN_COMMITMENT _TYPE	A assinatura foi criada usando um tipo de política e compromisso desconhecido para o SVA.
INVÁLIDA	TIMESTAMP_ORDER _FAILURE	Algumas restrições na ordem dos carimbos de data / hora de assinatura e / ou carimbos de data / hora do (s) objeto (s) de dados assinados não são respeitadas.
INVÁLIDA	GENERIC	Qualquer outro motivo
INDETERMINADA	NO_SIGNER_CERTIFICATE _FOUND	O certificado do signatário não pode ser identificado.
INDETERMINADA	NO_CERTIFICATE_CHAIN _FOUND	Nenhuma cadeia de certificados foi encontrada para o certificado do signatário identificado.
INDETERMINADA	REVOKED_NO _POE	O certificado do signatário foi revogado na validação Data/hora. No entanto, o Algoritmo de Validação de Assinatura não pode determinar se a hora de assinatura é anterior ou posterior à hora de revogação.
INDETERMINADA	REVOKED_CA _NO_POE	Foi encontrada pelo menos uma cadeia de certificados, mas foi descoberto que um certificado CA intermediário foi revogado.
INDETERMINADA	SIGNED_DATA_NOT _FOUND	Não é possível obter dados assinados.

Continua na próxima página

Tabela 2.2 – continuação da página anterior

Indicação do Resultado	Sub-indicação do Resultado	Significado
INDETERMINADA	OUT_OF_BOUNDS_NO _POE	O certificado do signatário expirou ou ainda não é válido na validação data / hora e o Algoritmo de Validação de Assinatura não podem determinar se a hora de assinatura está dentro do intervalo de validade do certificado do signatário.
INDETERMINADA	CRYPTO_CONSTRAINTS _FAILURE_NO_POE	Pelo menos um dos algoritmos que foram usados em um material (por exemplo, o valor da assinatura, um certificado ...) envolvido na validação da assinatura ou o tamanho das chaves usadas com tal algoritmo não é mais considerado confiável na data de validação /Tempo. No entanto, o Algoritmo de Validação de Assinatura não pode garantir que o material em questão foi produzido antes ou depois do algoritmo ou o tamanho das chaves foi considerado não confiável.
INDETERMINADA	NO_POE	Uma prova de existência está faltando para determinar que um objeto assinado foi produzido antes de algum evento comprometedor (por exemplo, algoritmo quebrado).
INDETERMINADA	NO_POLICY	A política a ser usada para validação não pôde ser identificada.

Continua na próxima página

Tabela 2.2 – continuação da página anterior

Indicação do Resultado	Sub-indicação do Resultado	Significado
INDETERMINADA	TRY_LATER	Nem todas as restrições podem ser atendidas usando as informações disponíveis. No entanto, pode ser possível fazer isso usando informações adicionais de revogação que estarão disponíveis em um momento posterior.
INDETERMINADA	GENERIC	Qualquer outro motivo.

2.3.2 eIDAS

Em 2014, foi introduzido através da Normativa (UE) n.º 910/2014, também conhecido como Normativa eIDAS, Identificação Eletrónica, Autenticação e Serviços de Confiança - Electronic Identification, Authentication and Trust Services (eIDAS), o esquema operacional que tem como objetivo a identificação e a proteção de interações eletrónicas entre empresas, cidadãos e entidades públicas dentro da União Européia, de forma a ampliar a utilização de serviços *online* pelos cidadãos, operadores económicos e administração pública, além de tornar os processos de autenticação, contratação e assinatura mais eficientes, de acordo com Gabinete Nacional de Segurança [14]. Conforme mencionado anteriormente, foi introduzido um único esquema que abrage a Identificação Eletrónica - Electronic Identification (eID) e os Serviços de Confiança - AS, de forma a oferecer diversos serviços, tais como ilustrado na Figura 2.15 e mencionado por Digital Single Market [23].

Figura 2.15: Serviços oferecidos pelo eIDAS.[14]

1. **eID:** Oferece às empresas a oportunidade de realizar controlos mais rígidos sobre a identidade de clientes e outras empresas;

2. **eTimestamp:** Certificação que um documento existia em dado período;

[13] Fonte: Adaptado de https://www.etsi.org/deliver/etsi_ts/102800_102899/102853/01.01.02_60/ts_102853v010102p.pdf

[14] Fonte: https://ec.europa.eu/futurium/sites/futurium/files/eidas-guidebooken.pdf

3. **eSignature:** Corresponde a expressão em formato eletrónico da concordância de uma entidade/pessoa com o conteúdo de um documento ou conjunto de dados. As assinaturas eletrónicas qualificadas possuem o mesmo efeito como assinaturas manuscritas;
4. **Certificado Qualificado de Autenticação Web - Qualified Web Authentication Certificate (QWAC):** Certificado eletrónico que prova que tal utilizador é dono de um *website*;
5. **eSeal:** Equivalente a um selo aplicado num documento que garante a sua origem e também a sua integridade;
6. **Serviço Eletrónico de Entregas Registadas - Electronic Registered Delivery Service (ERDS):** Serviço que permite a transmissão eletrónica de informações entre empresas, cidadãos e administração pública. Ele garante o envio, recebimento e protege contra o risco de perda, roubo, dano ou alterações não autorizadas.

2.3.2.1 Benefícios do eIDAS

O eIDAS apresenta inúmeros benefícios [23], tanto para as empresas, como para os utilizadores, a saber:

1. Redução na quantidade de papel, visto o processo ser eletrónico;
2. Processos de autenticação, assinatura e contratos mais eficientes;
3. Aumento da confiança do utilizador;
4. Verificação confiável dos registros financeiros e de identidade dos clientes;
5. Redução na burocracia com a utilização remota de assinaturas digitais, que são válidas da mesma forma que a assinatura manual;
6. Rastreamento de aprimorado de documentos;
7. Redução de encontros físicos para compartilhamento *online* seguro de acordos contratuais;
8. Selagem remota da documentação com autenticidade e integridade garantida da informação.

2.3.2.2 Níveis de Assinaturas

Neste tópico serão descritos os três níveis de assinaturas digitais definidas pela Diretiva 1999/93/EC similar ao eIDAS, segundo mencionado por Namirial [13].

- **Assinatura Eletrónica - Electronic Signature (ES):** Este tipo de assinatura é considerada apenas um princípio jurídico sem efeito probatório. Pode constituir uma assinatura eletrónica por exemplo, escrever o nome no fim do *e-mail*.

- **Assinatura Eletrónica Avançada - Advanced Electronic Signature (AES):** Este tipo de assinatura prova que um indivíduo assinou um documento num certo período, e deve seguir alguns requisitos, tais como:

  1. Deve estar vinculado ao ID do signatário de forma única e intransferível;
  2. Deve permitir a sua identificação;
  3. Deve ser criado utilizando dados que o signatário pode usar com confiança e ter controlo exclusivo;
  4. Deve invalidar a assinatura se os dados assinados forem alterados de alguma maneira.

  Através destes recursos, garante-se um nível alto de segurança nos processos de assinatura e de questões contratuais. Este tipo de assinatura difere da ES em relação a obrigatoriedade do ID do utilizador ser associado à assinatura, como demanda a lei para operações de risco.

- **Assinatura Eletrónica Qualificada - Qualified Electronic Signature (QES):** Este tipo de assinatura é uma versão da AES baseada num certificado qualificado, que é criado pelo Dispositivo de Criação de Assinaturas Eletrónicas Qualificadas - Qualified Electronic Signature Creation Device (QSCD), sendo esta a diferença. O certificado para este tipo de assinatura é emitido por um provedor de serviços de confiança qualificado que atesta a autenticidade da assinatura eletrónica. Para além disto, o QSCD tem o papel de garantir que somente o signatário possua o controlo da chave privada, que os dados de criação de assinatura são geridos por um provedor qualificado e que os dados de criação da assinatura são exclusivos, confidenciais e protegidos contra falsificações [4]. Com base no eIDAS, somente este tipo de assinatura possui explicitamente os mesmos efeitos legais que uma assinatura manual em todos os estados da União Européia. A Figura 2.16 ilustra os três níveis de assinaturas.

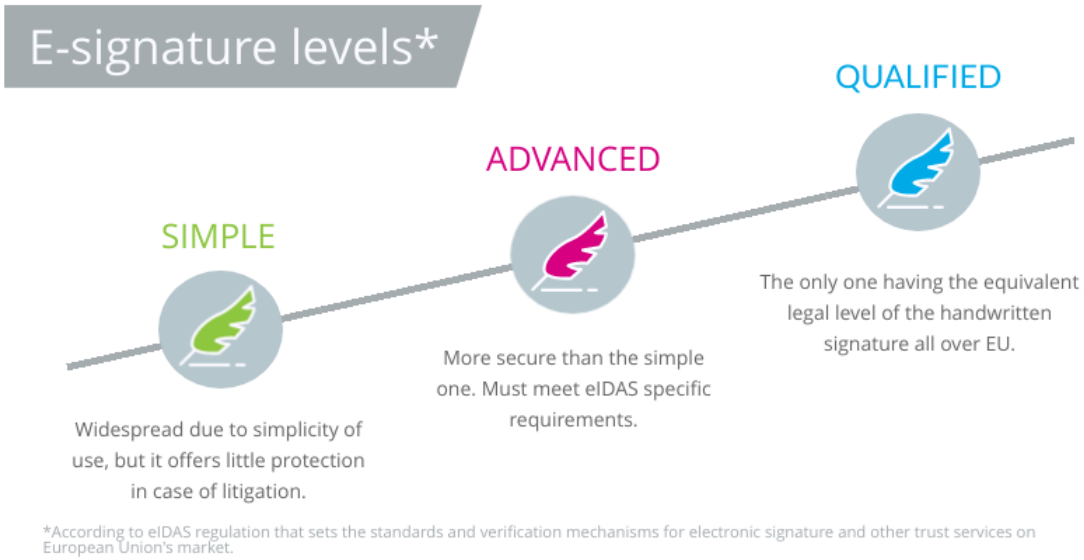


Figura 2.16: Os três níveis de assinatura segundo o eIDAS.[15]

[15]Fonte: https://www.luxtrust.com/e-signature-levels-and-legal-value/

2.3.2.3 Tecnologias para Implementação de Assinaturas Eletrónicas (eSignatures)

As assinaturas digitais podem ser classificadas em três tipos de formatos/perfis consoante o tipo de utilização, segundo Namirial [13], e para os quais o foco será no primeiro formato, PAdES.

1. **Assinatura Eletrónica Avançada do PDF - PDF Advanced Electronic Signatures (PAdES):** São um conjunto de extensões utilizadas em Documentos de Formato Portátil - Portable Document Format (PDF) que contém assinaturas eletrónicas. Sua característica principal é permitir a visualização eletrónica, geralmente por uma grande audiência, da mesma forma que a impressa, por isso é muito utilizada em empresas.
2. **Assinaturas Eletrónicas Avançadas do XML - XML Advanced Electronic Signatures (XAdES):** Destinam-se a assinar qualquer tipo de documento eletrónico, incluindo imagens, ficheiros de mídia, qualquer tipo de dados binários, documentos PDF e, em especial, XML. Além de permitir que vários documentos possam ser agrupados logicamente e assinados com um único XAdES ou paralelamente.
3. **Assinaturas Eletrónicas Avançadas do CMS - CMS Advanced Electronic Signatures (CAdES):** Destinam-se a assinar qualquer tipo de formato, binário ou texto, incluindo PDF. Entretanto, o *output* é um ficheiro em formato p7s (cifrado com assinaturas digitais), que não é de fácil visualização por leitores de ficheiros comuns. A saber, uma Sintaxe de Mensagem Cifrada - Cryptographic Message Syntax (CMS), é um padrão utilizado por esquemas criptográficos e protocolos para assinar, autenticar ou cifrar qualquer dado digital.

2.3.2.4 Assinatura Eletrónica Avançada em Ficheiros PDF (PAdES)

O PAdES, de acordo com o ETSI [10], é dividido em 4 perfis apresentados abaixo. Nesta dissertação o foco será somente nos Perfis Básico e de Longo Prazo.

1. **PAdES Perfil Básico** - Perfil baseado na Organização Internacional para Padronização - International Organization for Standardization (ISO) 32000-1;
2. **PAdES Melhorado (*Enhanced*)** - Perfis PAdES Basic Electronic Signature (PAdES-BES) e PAdES Explicit Policy Electronic Signature (PAdES-EPES);
3. **PAdES de Validação de Longo Prazo** - Perfil PAdES Long Term Validation (PAdES-LTV);
4. **PAdES para Conteúdos XML** - Perfis para assinaturas XAdES.

Alguns dos benefícios da utilização do PAdES para empresas e para utilizadores comuns são os seguintes, de acordo com Signicat [28]:

- Formato único;

- Pode ser distribuído à terceiros;
- Contém a evidência completa da assinatura;
- Permite que todas partes do acordo a possuam;
- Pode ser lida por qualquer pessoa com um leitor PDF;
- Consegue manter várias assinaturas em um mesmo documento;
- Utiliza um padrão aberto para criação e validação das assinaturas.

O Perfil Básico PAdES é um formato específico para assinaturas digitais em PDF que seguem as especificações da ISO 32000-1:2008, que permite que qualquer pessoa com um leitor de PDF possa: visualizar o que foi assinado, por quem e como foi assinado, além de permitir a sua validação em longo prazo. A Figura 2.17 exemplifica um ficheiro PDF assinado seguindo o PAdES.

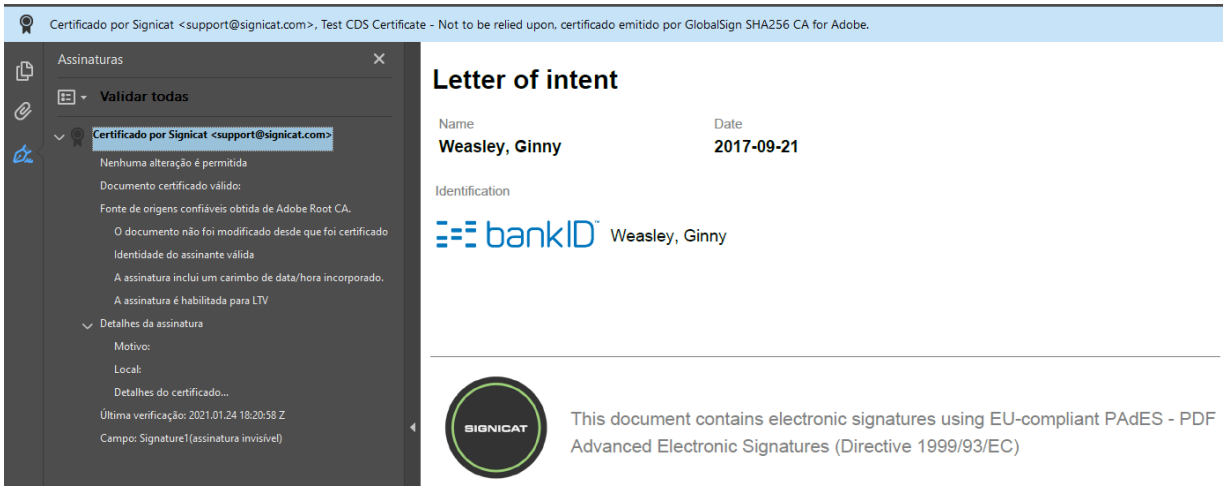


Figura 2.17: Exemplo de um ficheiro assinado seguindo o PAdES.[16]

A norma garante que todo *software* de visualização compatível, como Adobe Acrobat Reader, consiga carregar corretamente PDF assinados digitalmente sem nenhum *software* proprietário [13]. Seguindo a ISO 32000-1, o ficheiro PDF deve conter em sua estrutura de dados a assinatura com várias outras informações opcionais, de forma que o valor da assinatura é codificado como um objeto binário. Esta estrutura é conhecida como Dicionário de Assinaturas, de acordo com ETSI [10] e ilustrado na Figura 2.18.

Por outro lado, o perfil PAdES-LTV é utilizado para solucionar o problema de validação de assinatura de documentos que são armazenados por um longo período de tempo, e que sejam necessários verificar as assinaturas muito depois destas terem sido criadas ou após o certificado da assinatura ter expirado.

[16]Fonte: Editado de https://developer.signicat.com/wp-content/uploads/2017/09/pades_example-1.pdf
[17]Fonte: https://www.etsi.org/deliver/etsi_ts/102700_102799/10277801/01.01.01_60/ts_10277801v010101p.pdf

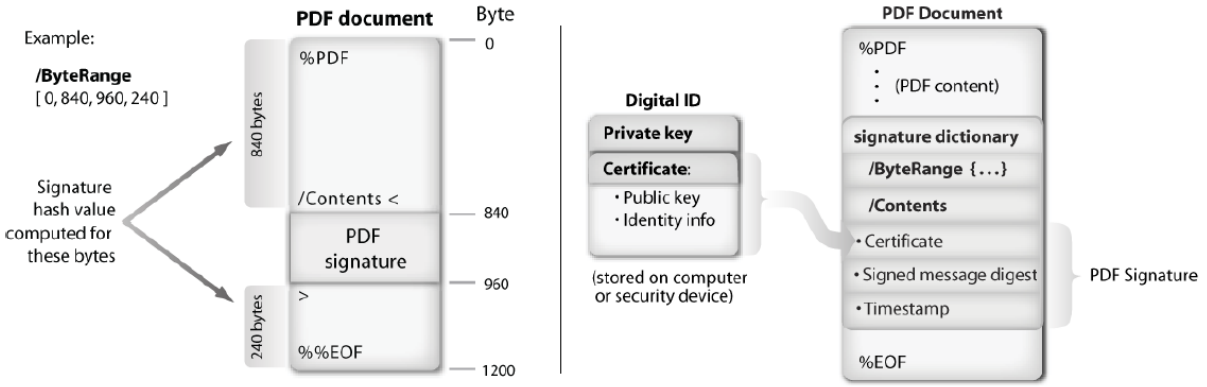


Figura 2.18: O valor binário da assinatura é colocado no conteúdo (contents) do dicionário de assinatura.[17]

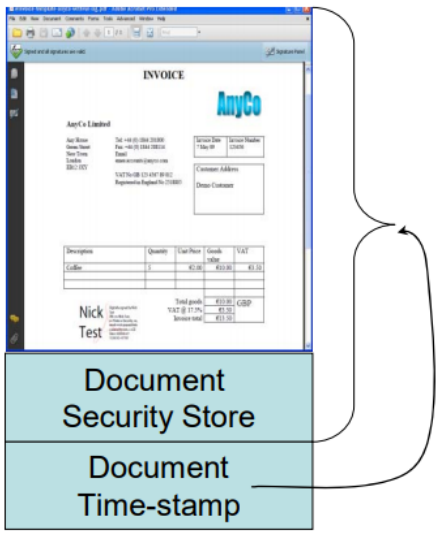


Figura 2.19: Ficheiro PDF com perfil LTV.[18]

Os dados originais de validação podem não estar mais disponíveis, além da proteção criptográfica conferida pela assinatura não ser garantida depois que o certificado esteja expirado. O problema é solucionado através da adição de uma marca temporal que protege o documento existente e quaisquer dados de validação [11], Figura 2.19.

Para além disto, o Perfil PAdES-LTV utiliza uma extensão para ISO 32000-1 chamada Loja de Documentos de Segurança - Document Security Store (DSS), utilizada para transportar toda informação necessária para validação de assinaturas, e a Marca Temporal do Documento (*Document Time-stamp*) que é utilizado para estender o tempo de vida de proteção ao documento. E, opcionalmente, pode-se utilizar as Informações Relacionadas à Validação - Validation Related Information (VRI), que trata de correlacionar uma assinatura específica com os dados de validação [11]. A Figura 2.20 ilustra a estrutura de um DSS e um VRI.

[18]Fonte: https://www.etsi.org/deliver/etsi_ts/102700_102799/10277804/01.01.01_60/ts_10277804v010101p.pdf

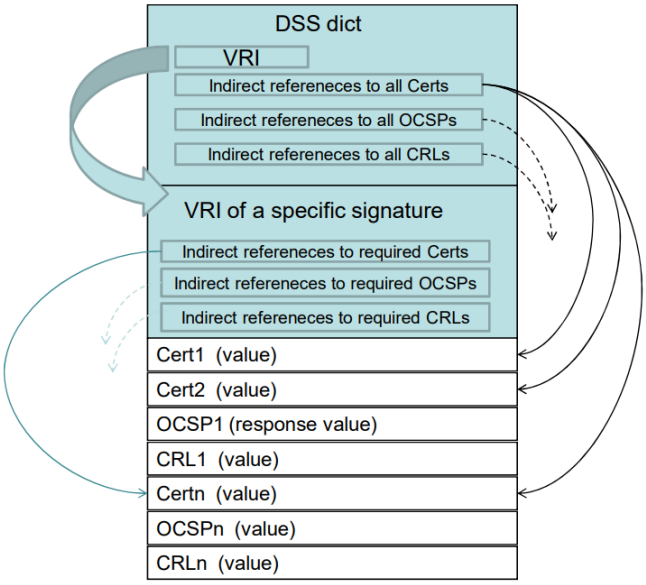


Figura 2.20: Estrutura de um DSS e VRI.[2.19]

2.4 Desenvolvimento Web

O desenvolvimento Web pode ser definido como o processo de criação de um *website* para a Internet ou uma Intranet (uma rede privada). O ***front-end*** é a interface gráfica para o utilizador, na qual o comportamento e recursos visuais são executados no navegador, enquanto o ***back-end*** corresponde a parte dos servidores e *scripts*, que realizam e manipulam os dados solicitados pelo cliente. De forma geral, conforme ilustrado na Figura 2.21, o triângulo da programação Web é composto por: HTML, Folhas de Estilo em Cascata - Cascading Style Sheets (CSS), e Javascript. O HTML é responsável por definir os conteúdos das páginas, o JavaScript é utilizado para programar o comportamento das páginas, e o CSS é responsável por especificar o *layout* das páginas.

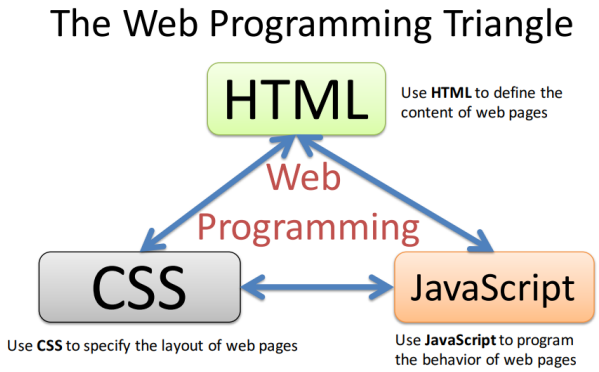


Figura 2.21: Triângulo de programação web.[19]

O funcionamento básico de funcionamento de um cliente-servidor, de acordo com [16], e

[19]Fonte: https://www.multitech.ac.ug/uploads/Introduction%20to%20Web%20Programming.pdf

ilustrado na Figura 2.22, se dá através da seguinte forma:

1. O cliente realiza um pedido através do navegador;
2. O navegador envia o pedido para o servidor;
3. O servidor realiza a busca da informação solicitada na base da dados;
4. O cliente visualiza a informação solicitada.

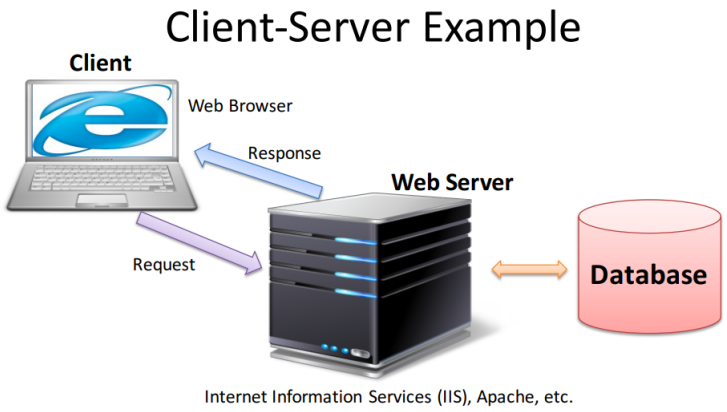


Figura 2.22: Funcionamento básico de um cliente-servidor[2.21]

2.4.1 REST API

Fundalmentalmente, o Estado Representacional de Transferência - Representational State Transfer (REST) é um modelo de arquitetura que fornece interações distribuídas entre sistemas e navegadores em qualquer dispositivo, servidor, aplicativo ou banco de dados, conforme menciona Hans-Petter Halvorsen [29]. É utilizado para desenvolver sistemas rápidos, escaláveis, leves e fáceis de serem mantidos. O seu modo básico de funcionamento é realizado através das seguintes ações: *GET, PATCH, POST, PUT* e *DELETE* [20].

- *GET:* Recuperar um recurso existente;
- *PATCH:* Aplicar modificações parciais em um recurso;
- *POST:* Criar uma nova entrada de um recurso;
- *PUT*: Modificar um recurso existente;
- *DELETE:* Remover um recurso existente.

Toda resposta de uma REST API contém um código de resultado no cabeçalho do pedido, que indica se o pedido foi bem sucedido ou não. A Tabela 2.3 ilustra alguns dos principais *status code*.

Tabela 2.3: Códigos de resultados mais comuns.

Códigos de Resultado		
Status Code	Nome	Descrição
200	OK	Pedido bem sucedida
201	Created	Pedido bem sucedida e um novo recurso foi criado
302	Found	URI do recurso requerido foi mudada temporariamente
400	Bad Request	Erro de sintaxe do pedido
401	Unauthorized	Erro de autorização
403	Forbidden	O utilizador não possui direitos de acesso
404	Not Found	O recurso solicitado não foi encontrado
500	Internal Server Error	Erro interno
503	Service Unavailable	O serviço REST está temporariamente indisponível

No corpo da resposta do pedido podem existir inúmeras coisas, como por exemplo: informações, HTML, imagens, ficheiro de áudio, entre outras. As respostas normalmente são codificadas em JSON, entretanto outros formatos como: Linguagem Extensível de Marcação Genérica - Extensible Markup Language (XML), Valores Separados por Vírgulas - Comma-separated values (CSV), ou *strings* simples podem ser utilizados.

2.4.1.1 Segurança

Todo serviço REST deve ser desenhado de forma que exista autenticação, e que somente o utilizador que tenha a devida permissão possa aceder ao conteúdo. Na Figura 2.23, pode-se observar um pedido e uma resposta a um utilizador que não tem permissões de aceder a um certo conteúdo. Como resposta, é retornado o *status code* 401, que indica que ele não possui autorização e o acesso é recusado.

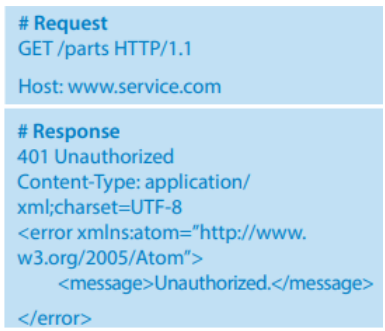


Figura 2.23: Exemplo de um pedido REST.[20]

Uma abordagem comum é atribuir aos utilizadores uma ou mais funções e, em seguida, conceder segurança a essas funções. O sistema deve ser capaz de definir controlo de acesso à essas funções em vários níveis, como assistentes, diretores, administradores, etc.

[20]Fonte: https://www.infosys.com/digital/insights/documents/restful-web-services.pdf

Para além disto, no que tange o desenvolvimento seguro, existe o guia da Open Web Application Security Project (OWASP) que fornece aos desenvolvedores, arquitetos e especialistas uma lista de recomendações, estratégias e soluções para compreender e mitigar vulnerabilidades únicas e riscos de segurança que possam surgir numa API. A Figura 2.24 destaca as vulnerabilidades mais comuns numa API.

Figura 2.24: Vulnerabilidades mais comuns numa API.[21]

2.4.1.2 Autenticação

A autenticação numa REST API pode ser realizada de várias formas, as mais conhecidas são através do Protocolo de Transferência de Hipertexto - Hypertext Transfer Protocol (HTTP) Basic Authentication, Protocolo de Acesso a Diretório Leve - Lightweight Directory Access Protocol (LDAP), Autorização Aberta 2.0 - Open Authorization (Oauth) 2.0, *API KEYS* e JSON Web Tokens (JWT). De forma geral, funcionam através de *cookies*, que armazenam os *tokens* de sessão dos utilizadores. A seguir é apresentado brevemente uma descrição de cada forma de autenticação.

1. **HTTP Basic Authentication:** Este tipo de autenticação é composto pelo utilizador e senha codificados em Base64, Base64(username:password), enviados no cabeçalho do pedido, Figura 2.25.

2. **LDAP:** O processo de autenticação LDAP é realizado através da validação do utilizador e *password* numa serviço de diretório como MS Active Directory, OpenLDAP ou OpenDJ, por exemplo. Se as credenciais estiverem corretas, o servidor de diretório retornará sucesso, Figura 2.26. Caso contrário, ele retorna um erro de LDAP - Credenciais inválidas.

[21] Fonte: https://www.slideshare.net/42crunch/owasp-api-security-top-10-api-world-182516159
[22] Fonte: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Authentication
[23] Fonte: https://doubleoctopus.com/security-wiki/protocol/lightweight-directory-access-protocol/

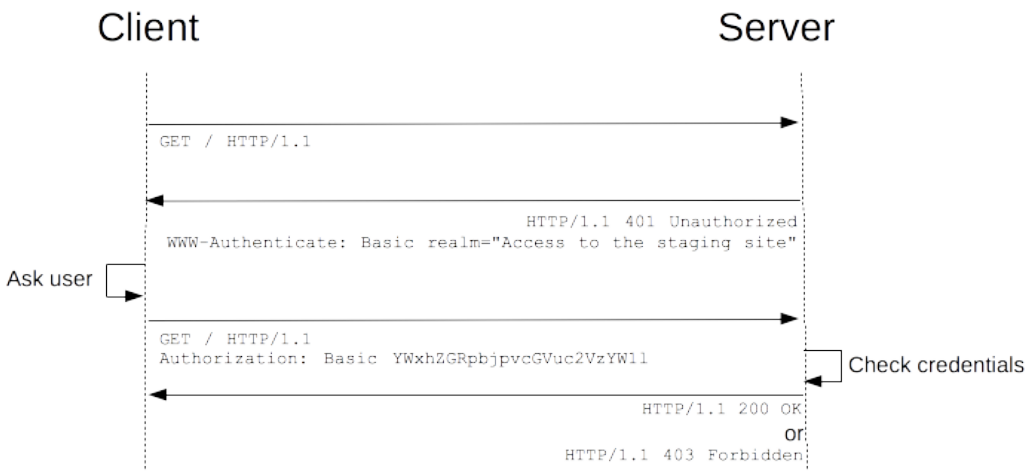


Figura 2.25: Exemplo de um pedido HTTP Basic.[22]

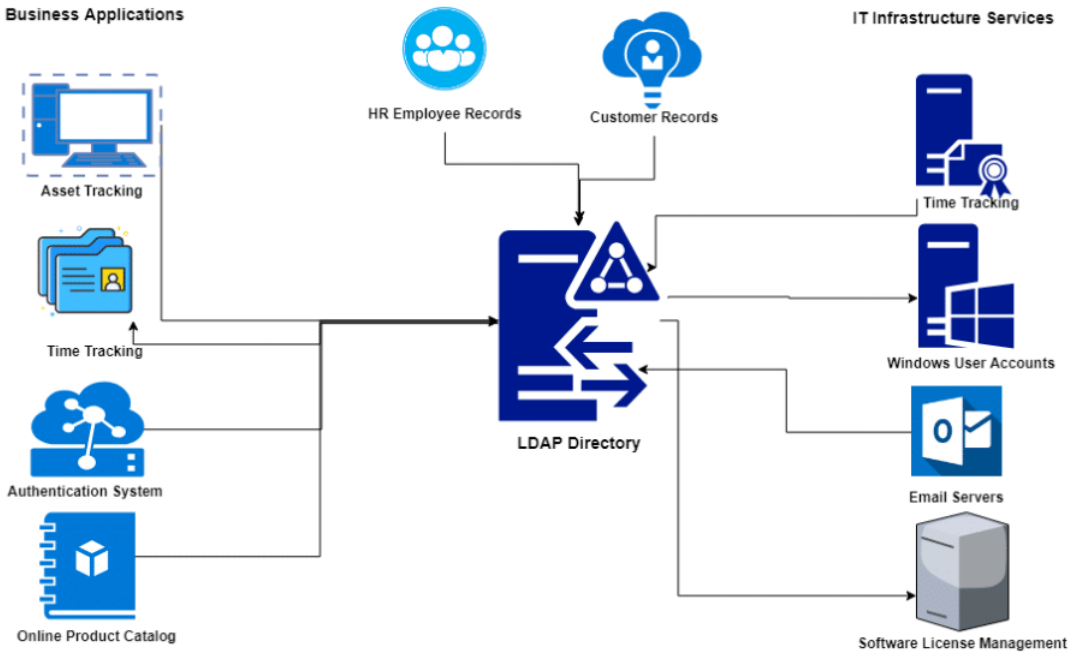


Figura 2.26: Exemplo de um pedido LDAP.[23]

3. **OAuth 2.0:** Antes de qualquer pedido ser realizado, é enviado o ID do utilizador e um segredo para o servidor OAuth, que verifica e gera um *token* que é enviado de volta para o cliente para ser utilizado em cada pedido, até que o mesmo expire, Figura 2.27.

4. **API Keys:** Este tipo de autenticação funciona através da geração de uma chave que dá direitos específicos à um utilizador. O utilizador realiza o *login* no portal de serviço e cria uma chave de API, esta chave é enviada em todo pedido posterior, Figura 2.28.

[24]Fonte: https://docs.oracle.com/cd/E82085_01/160027/JOS%20Implementation%20Guide/Output/oauth.htm

[25]Fonte: https://blogs.oracle.com/integration/api-key-based-authentication:-quickly-and-easily

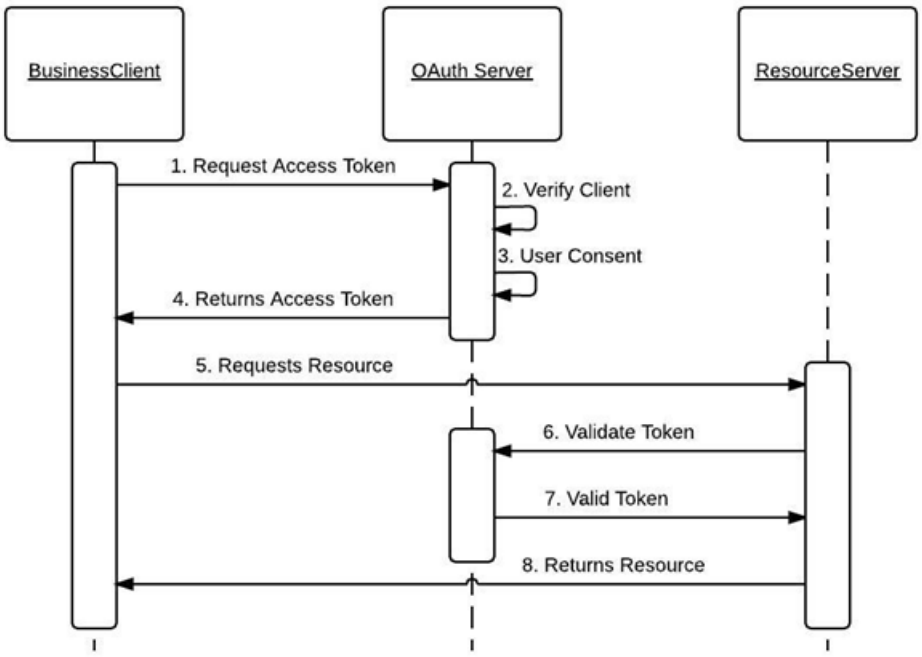


Figura 2.27: Fluxo de autenticação OAuth 2.0.[24]

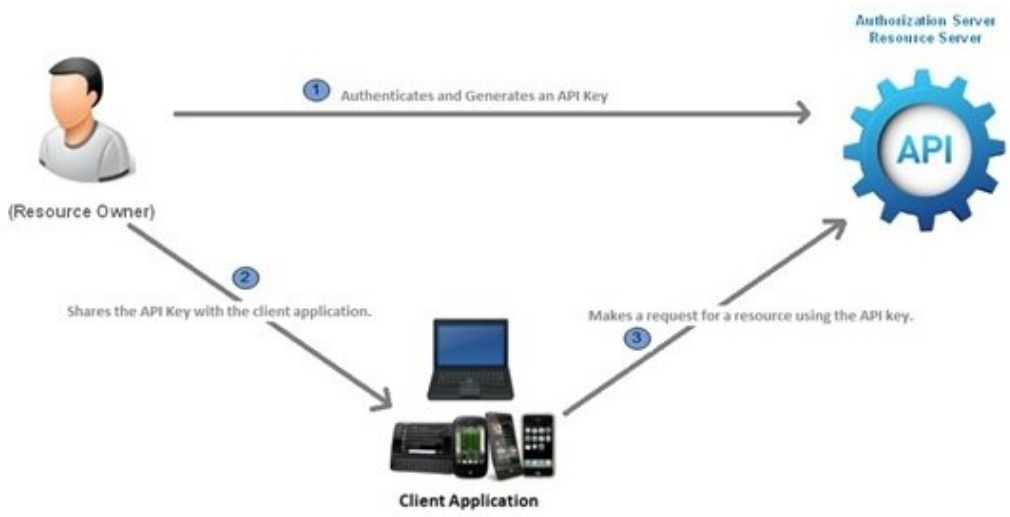


Figura 2.28: Fluxo de autenticação com API keys.[25]

5. **JWT:** É um método de autenticação baseado em *tokens* composto por três partes: o cabeçalho, o *payload* e a assinatura. O cabeçalho e o *payload* são assinados digitalmente e são separados por um ".", Figura 2.29.

[26]Fonte: https://medium.com/@sureshdsk/how-json-web-token-jwt-authentication-works-585c4f076033

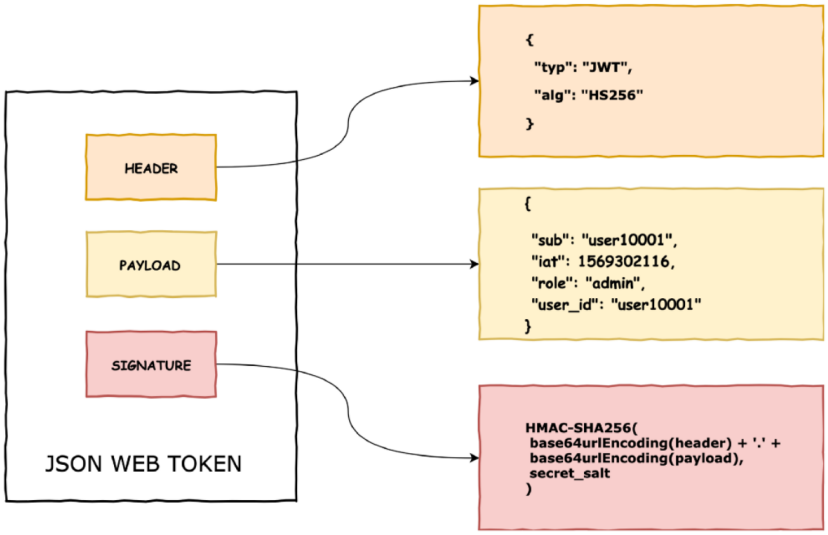


Figura 2.29: Partes de um token JWT.[26]

Quando o utilizador fornece as credenciais corretas para um terminal de *login*, o servidor cria um *token* e o retorna na resposta. Este *token* é enviado tanto no cabeçalho do pedido, como no da resposta, Figura 2.30.

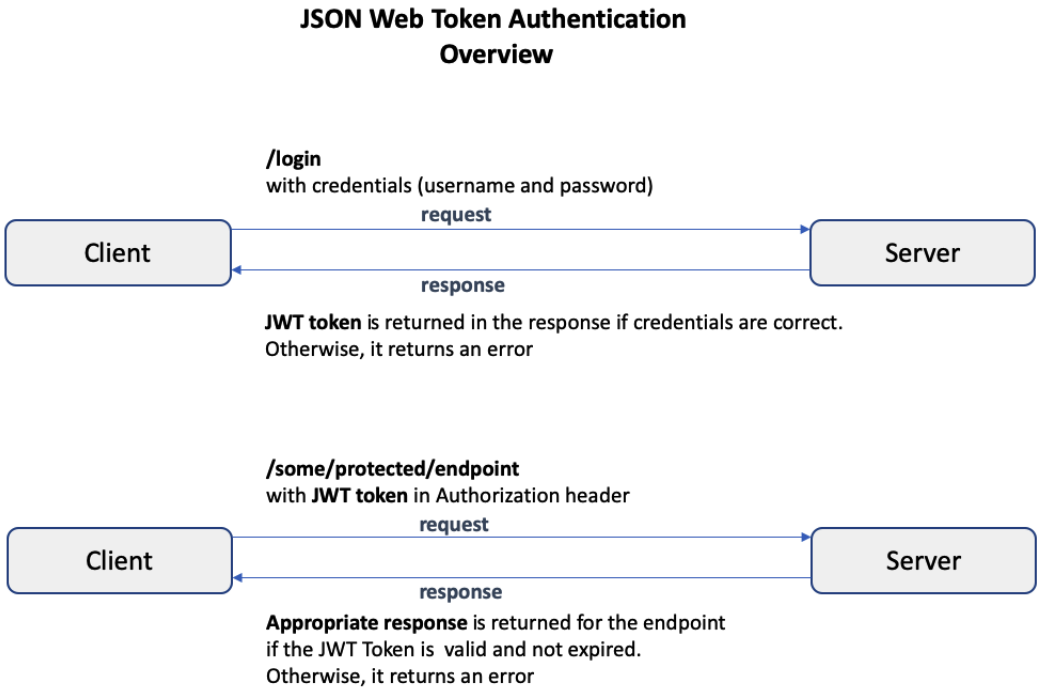


Figura 2.30: Fluxo de autenticação com JWT.[27]

[27] Fonte: https://loopback.io/doc/en/lb4/Authentication-tutorial.html

2.4.2 Spring Boot

O *Spring Boot* é uma estrutura de código-aberto escrita em Java, utilizada para criar microserviços (arquitetura que permite aos desenvolvedores criarem e implementarem serviços de forma independente). Cada serviço em execução tem seu próprio processo e isso permite que o modelo seja leve e ofereça suporte para aplicações de negócios.

O *Spring Boot* também atua com a gestão de dependências de uma aplicação de forma automatizada, afim de simplificar a execução do projeto em tempo de desenvolvimento e depuração. Além disto, o *Spring Boot* possui a funcionalidade de empacotar a aplicação num ficheiro `.JAR` executável que contém todas as dependências necessárias, incluindo o servidor Web Java (*Servlet Container*), como o Tomcat ou Jetty. Algumas das vantagens do *Spring Boot* para os desenvolvedores, de acordo com Anatoly Mihalchenko [26] são:

- Fácil de entender e desenvolver aplicações;
- Aumento da produtividade;
- Diminuição do tempo de desenvolvimento;
- Ajuda a autoconfigurar todos os componentes para uma aplicação;
- Possui servidores HTTP embutidos, como Jetty e Tomcat.

2.4.2.1 Arquitetura de Fluxo do Spring Boot

O *Spring Boot* possui quatro camadas:

1. Camada de Apresentação (*Presentation Layer*): Consiste em visualizações, ou seja, a parte do *front-end*;
2. Camada de Acesso a Dados (*Data Access Layer*): Operações no banco de dados, CRUD (criar, recuperar, atualizar e excluir);
3. Camada de Serviço (*Service Layer*): Consiste em classes de serviço e utiliza serviços fornecidos pela Camada de Acesso a Dados.
4. Camada de Integração (*Integration Layer*): Consiste em diferentes serviços *Web* (qualquer serviço disponível na Internet e que utiliza o sistema de mensagens XML).

O fluxo da arquitetura do *Spring Boot* pode ser observada na Figura 2.31 e acontece da seguinte forma:

- Um cliente realiza uma solicitação HTTPS;

[28] Fonte: https://www.geeksforgeeks.org/introduction-to-spring-boot/

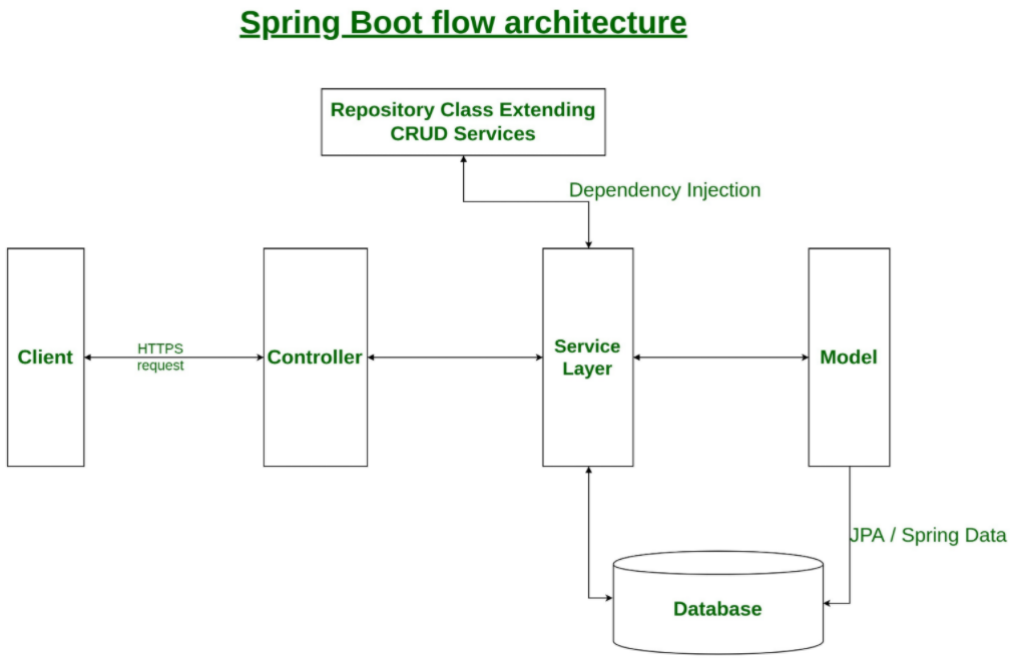


Figura 2.31: Arquitetura de fluxo do Spring Boot.[28]

- Em seguida, ela vai para o Controlador (*Controller*) e é mapeada com a rota conforme a solicitação e é tratada. Logo em seguida, a lógica de serviço é chamada, se necessária;
- A lógica de negócios é executada na Camada de Serviço (*Service Layer*) que também pode estar executando a lógica nos dados do banco de dados que são mapeados por meio da API Java Persistente - Java Persistence API (JPA), com modelo/classe de entidade;
- Por fim, uma Página JavaServer - JavaServer Pages (JSP), é retornada na resposta se nenhum erro ocorrer.

Capítulo 3

Estado da Arte

Existem atualmente diversos programas e *websites* que permitem validar assinaturas digitais, como o Adobe Reader, Ascertia, Secured Signing, DocsVault, entre outros. Apesar de boa parte deles conseguirem realizar as validações nas assinaturas, alguns deles constituem serviços comerciais pagos, como é o caso do Secured Signing. Para além disto podem-se elencar outras "limitações", como a necessidade de criar conta para utilizar, realização de configurações, possuirem uma lista de Certificados de Autoridade - Certification Authority (CA) muito reduzida, o que não permite validar a assinatura corretamente, e por fim não darem resultados de forma amigável e direta. Todos estes fatores de usabilidade e custo acabam por afastar os utilizadores desta tecnologia que pode facilitar e economizar tempo no dia a dia.

Uma solução mais adequada que foi encontrada para validações de assinaturas digitais é a da utilização da biblioteca *Digital Signature Service* (DSS) da Comissão Européia, na qual a página de teste está acessível em https://ec.europa.eu/cefdigital/DSS/webapp-demo/home. Apesar desta biblioteca também não produzir de forma clara e "amigável" os resultados para o utilizador. Não obstante, a biblioteca possui uma documentação bastante acessível, o que permite ao desenvolvedor realizar modificações no código, de acordo com suas necessidades de utilização. Atualmente a biblioteca possui suporte para diversos formatos e perfis de assinatura. E, para este trabalho, como definido pela Loqr, será utilizado o PAdES, visto que é o perfil com foco em ficheiros de formato PDF e o mais utilizado atualmente.

Este projeto tem como objetivo principal permitir que utilizadores possam validar as assinaturas digitais contidas em documentos do formato PDF através do *website* da Loqr. Desta forma, até mesmo um utilizador sem muita experiência informática pode validar e distinguir com clareza se o ficheiro submetido possui assinaturas digitais válidas ou não. Para além disto, o sistema vai produzir um relatório completo de todas as validações executadas, para o caso de um utilizador mais avançado que o queira consultar. Neste cenário, a implementação deste sistema de validação de assinaturas digitais no portal da Loqr trás visibilidade para a empresa e confiança e segurança por parte dos utilizadores, tendo a Loqr como referência de portal de validação.

Capítulo 4

Desenvolvimento

4.1 Análise de Requisitos

4.1.1 Requisitos Funcionais

Após reuniões com a Loqr, para este projeto ficaram definidos os seguintes requisitos funcionais:

- **RF1 - Submissão de Ficheiros:** O sistema deve aceitar a submissão de ficheiros em formato PDF.
- **RF2 - Validação de Assinatura Digital:** O sistema deve realizar a análise de uma ou múltiplas assinaturas digitais presentes no ficheiro submetido.
- **RF3 - Resultado Básico da Validação:** O sistema deve fornecer o resultado da análise da assinatura digital através de ícones e cores.
- **RF4 - Resultado Avançado da Validação:** O sistema deve fornecer um relatório detalhado de todo o processo de validação da análise da assinatura digital.

4.1.2 Requisitos Não Funcionais

4.1.2.1 Requisitos de Usabilidade

Os requisitos de usabilidade são atributos de qualidade, através da interação entre o sistema e o utilizador. Este requisito baseia-se no princípio que o sistema deve ser fácil de aprender e utilizar, de forma que a realização de qualquer tarefa seja agradável.

- **RU1 - Identidade Visual e Interface Amigável:** O sistema deve possuir uma linguagem simples e fácil de compreender, de forma que um utilizador sem experiência consiga utilizá-lo intuitivamente.

- **RU2 - Utilização de cores:** O sistema deve utilizar ícones com cores para indicar se a assinatura digital é válida, inválida ou indeterminada.
- **RU3 - Drag and Drop:** A interface *Web* deve permitir o utilizador arrastar e soltar, numa área específica, o ficheiro a ser analisado.
- **RU4 - Ambiente de Utilização:** O sistema deve ser compatível com os *browsers* Google Chrome, Mozilla Firefox e Microsoft Edge.
- **RU5 - Idioma:** O sistema deve ser escrito em Português.

4.1.2.2 Requisitos de Confiabilidade

Os requisitos de confiabilidade são aqueles que correspondem a probabilidade do sistema não causar nenhuma falha após certo período de tempo a realizar determinada tarefa, entre elas a disponibilidade, na qual um sistema com taxa de 999/1000, significa que a cada 1000 solicitações, 999 são atendidas.

- **RC1 - Disponibilidade do Sistema:** O portal de validação, bem como o sistema de validação devem estar disponível 24/7, exceto em caso de manutenção.
- **RC2 - Validação da Integridade Ficheiro:** O sistema deve validar se o ficheiro PDF não está corrompido.
- **RC3 - Validação da Existência de Assinatura no Ficheiro:** O sistema deve validar se existe pelo menos uma assinatura digital no ficheiro PDF submetido.
- **RC4 - Tamanho Máximo de Ficheiros:** O tamanho máximo de submissão de ficheiros deve ser de 100 *megabytes*.

4.1.2.3 Requisitos de Portabilidade

Os requisitos de portabilidade podem ser definidos como a facilidade do sistema ser transferido e executado para diferentes tipos de ambientes, como por exemplo, Windows, Linux, entre outros. Se o esforço para portabilidade for menor que o tempo gasto para o desenvolvimento, é dito que o sistema é portável.

- **RP1 - Portabilidade:** O sistema de validação será instalado num ambiente de virtualização (*Docker*).
- **RP2 - Compatibildade de Bibliotecas:** As bibliotecas utilizadas devem ser compatíveis com ambientes *Windows* e Linux.

4.1.2.4 Requisitos de Segurança

Os requisitos de segurança são aqueles que caracterizam que não serão permitidos acessos não autorizados ao sistema e aos dados, de forma a assegurar que a integridade seja mantida em caso de ataques ou acidentes.

- **RS1 - Integridade da comunicação:** Toda comunicação entre o *browser* e o servidor deve ser feito de forma segura.
- **RS2 - Validação do ficheiro enviado:** Todo ficheiro submetido pelo utilizador deve ser pré-processado antes de qualquer validação no servidor. O reconhecimento será através do cabeçalho do ficheiro.
- **RS3 - Apagar Ficheiro PDF:** Todo ficheiro PDF deve ser apagado do servidor após a validação da assinatura digital.
- **RS4 - Bibliotecas atualizadas:** As bibliotecas criptográficas utilizadas pelo sistema devem estar atualizadas de forma a eliminar possíveis vulnerabilidades.

4.2 Arquitetura

4.2.1 Estrutura Alto Nível

Conforme a Figura 4.1 ilustra, o processo de funcionamento se inicia com a submissão pelo utilizador de um ficheiro PDF na interface. De seguida, o ficheiro é analisado pelo *back-end* da aplicação composto por um Docker executando o serviço do DSS *Bundle*. Como resultado, a(as) assinatura(as) no PDF são classificadas como: inválidas, indeterminadas ou válidas. Caso o ficheiro submetido seja inválido ou não contenha assinatura, também é indicado ao utilizador através de uma tela personalizada de erro. Um relatório com informações básicas (indicação da assinatura, nome do documento, assinado por, tipo de assinatura, data da assinatura, etc.) também é apresentado. Para além disto, o utilizador pode descarregar o relatório básico, assim como o relatório avançado com todas as informações do processo de validação.

4.2.2 Diagrama de Blocos do Processo de Validação DSS

De acordo com a Comissão Europeia [3], o processo de validação da biblioteca DSS segue a normativa mais atual do Instituto Europeu de Normas de Telecomunicações - European Telecommunications Standards Institute (ETSI), que é responsável por apoiar o desenvolvimento de produção de normas de telecomunicações, atividades de pré-normalização e normalização nas áreas das tecnologias da informação [2].

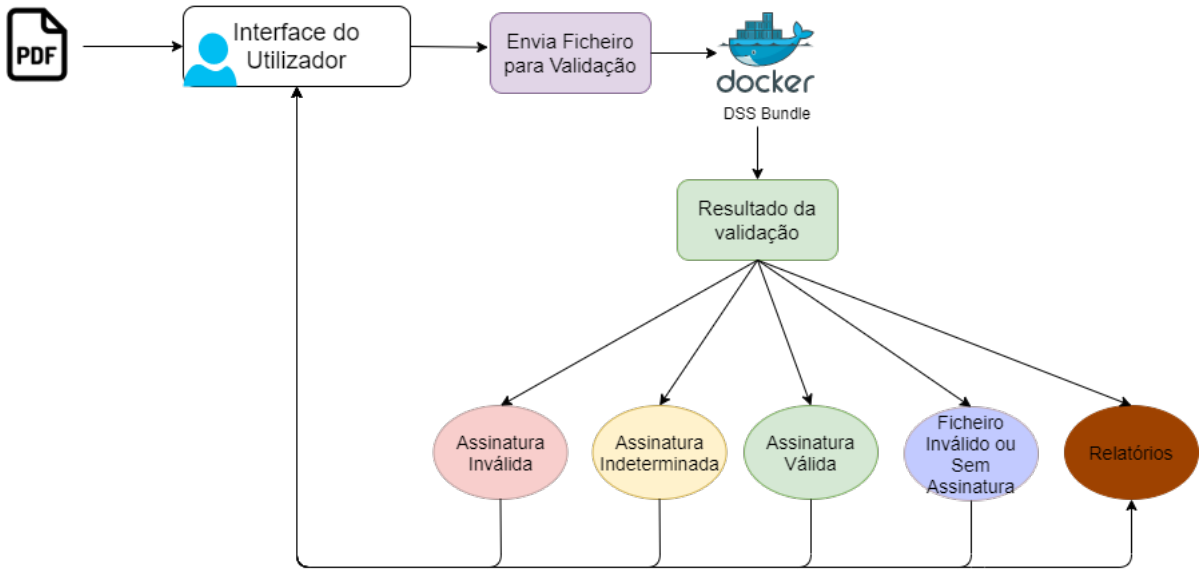


Figura 4.1: Estrutura alto nível do funcionamento da aplicação.

A Figura 4.2 ilustra o processo de validação da assinatura digital pela biblioteca do DSS através de um diagrama de blocos simplificado que representa um conjunto lógico de verificações utilizadas no processo de validação. O processo de validação é orientado pelas políticas de validação e também permite a validação de assinaturas de longo prazo/termo, além de verificar a integridade dos dados através de verificações criptográficas [3].

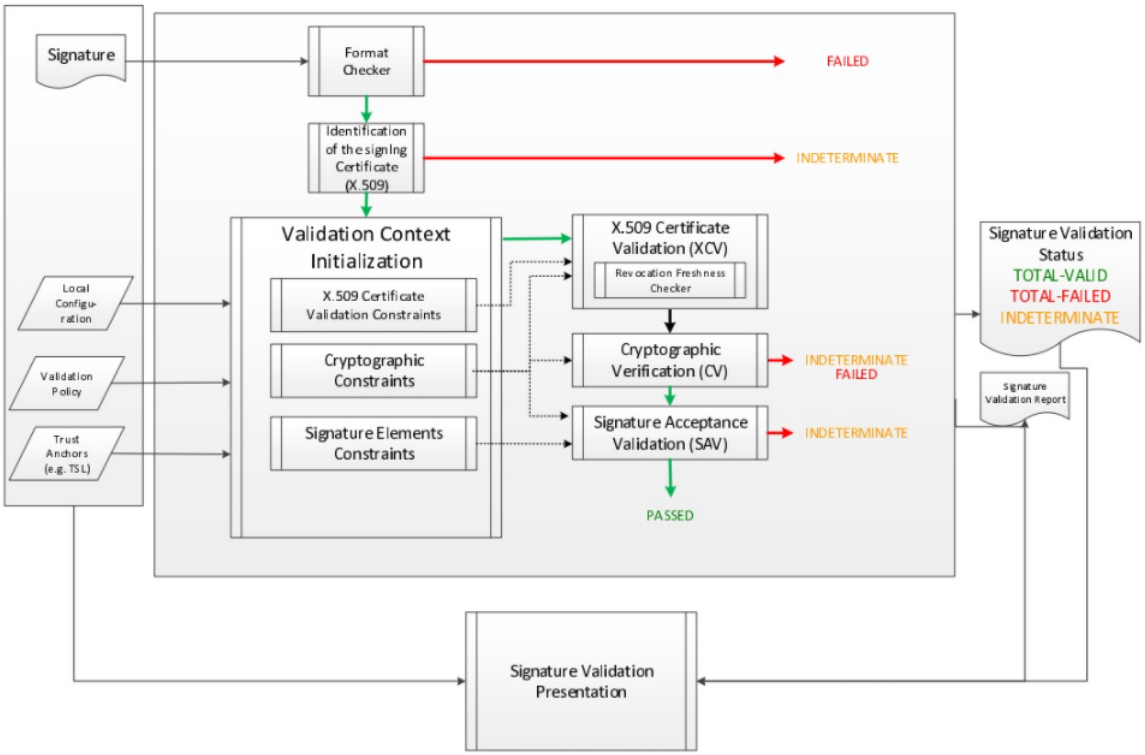


Figura 4.2: Processo de validação da biblioteca DSS.[1]

No fim da validação quatro tipos de relatórios são gerados: visão macroscópica, microscópica, dos dados de entrada e o relatório de validação ETSI em conformidade com a normativa, além de indicar se as assinaturas estão válidas. Cada um destes relatórios contém os diferentes níveis de detalhe relativos ao resultado da validação.

4.2.3 Integração entre os Componentes (MVC)

A integração entre os componentes da aplicação segue o padrão de projetos de *softwares* Modelo-Vista-Controlador - Model-View-Controller (MVC), que se desenrola da seguinte forma:

1. Ao enviar o ficheiro PDF para validação, é realizado um pedido HTTP para o Controlador;
2. O Controlador é responsável por direcionar a rota do pedido para o serviço DSS executando no Docker no Modelo;
3. Após o processamento do ficheiro, as informações da validação são enviadas para o Controlador;
4. O Controlador reencaminha para a Vista as informações HTML que serão apresentadas na interface do utilizador.

A integração entre os componentes pode ser observado na Figura 4.3 abaixo.

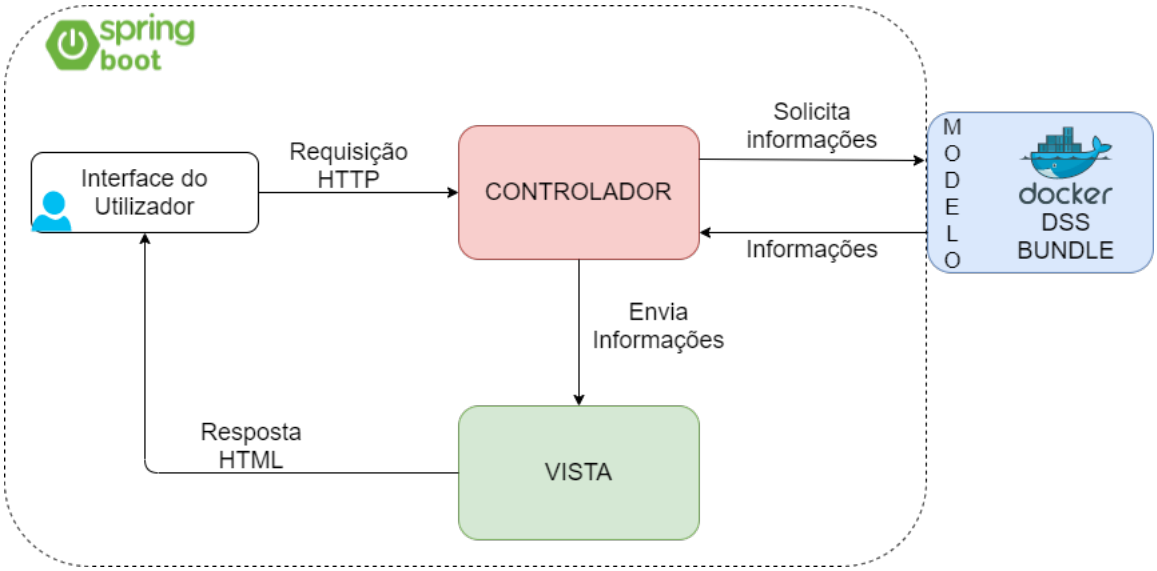


Figura 4.3: Arquitetura MVC com o DSS.

[1]Fonte: https://ec.europa.eu/cefdigital/DSS/webapp-demo/doc/dss-documentation.html#_validation_process

4.2.4 Diagrama da Sequência de Funcionamento e Obtenção dos Resultados

A Figura 4.4 representa o diagrama de sequência de funcionamento da aplicação. É possível observar o que cada componente da arquitetura é responsável por realizar durante todo o processo de validação, desde o carregamento da página principal até a obtenção dos resultados.

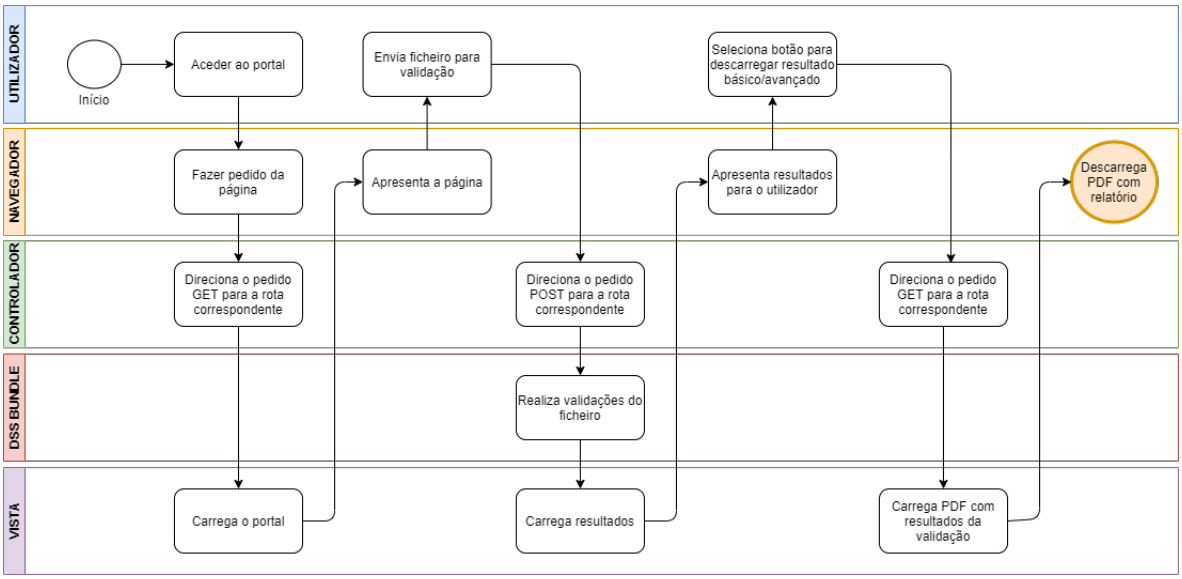


Figura 4.4: Diagrama de sequência de funcionamento e obtenção dos resultados.

Cada etapa do processo de funcionamento é descrita abaixo:

1. **Início:** Estado inicial;
2. **Aceder ao portal:** Utilizador digita o Localizador Padrão de Recursos - Uniform Resource Locator (URL), da aplicação no navegador;
3. **Fazer pedido da página:** Navegador solicita a página inicial;
4. **Direciona o pedido GET para a rota correspondente:** Controlador direciona o pedido;
5. **Carrega o portal:** Elementos da página HTTP são elaborados;
6. **Apresenta a página:** Página inicial da aplicação é carregada para o utilizador;
7. **Envia ficheiro para validação:** Utilizador envia ficheiro para validação;
8. **Direciona o pedido POST para a rota correspondente:** Controlador direciona o pedido de validação para o DSS *Bundle* (Modelo);
9. **Realiza validações do ficheiro:** DSS *Bundle* (Modelo) realiza todas as validações;
10. **Carrega resultados:** Os resultados são filtrados para exibição;

11. **Apresenta resultados para o utilizador:** Os resultados são exibidos para o utilizador;
12. **Seleciona botão para descarregar resultado básico ou avançado:** Utilizador escolhe qual relatório (básico ou avançado) de validação quer descarregar;
13. **Direciona o pedido GET para a rota correspondente:** Controlador direciona o pedido;
14. **Carrega resultados da validação:** Resultados da validação são carregados;
15. **Descarrega ficheiro com relatório:** Ficheiro com resultados é descarregado para utilizador.

4.2.5 MVC - Modelo | DSS Bundle

Conforme mencionado anteriormente, o processo de validação dos ficheiros submetidos pelo utilizador acontece na estrutura do Modelo, no qual está a executar uma imagem *Docker* do *DSS Bundle*. Primeiramente, é necessário garantir que o *software* Maven está instalado no computador, o *download* pode ser realizado através do seguinte *link*: https://maven.apache.org/download.cgi. Este *software* é utilizado compilar o código e para gerar o ficheiro que será utilizado no *Docker*. O código para o DSS Bundle utilizado para a validação é disponibilizado em https://github.com/esig/dss-demonstrations, ou diretamente no *site* da CEF Digital, https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/DSS.

A Figura 4.5 destaca o comando utilizado para compilar o código do DSS Bundle. Basicamente, este comando excluirá todos os ficheiros e recursos `Java .class` compilados anteriormente (como `.properties`) do projeto e a construção começará do zero. Por fim, irá então compilar, testar e empacotar o projeto Java. É importante ressaltar que o comando deve ser executado dentro da pasta raiz do projeto.

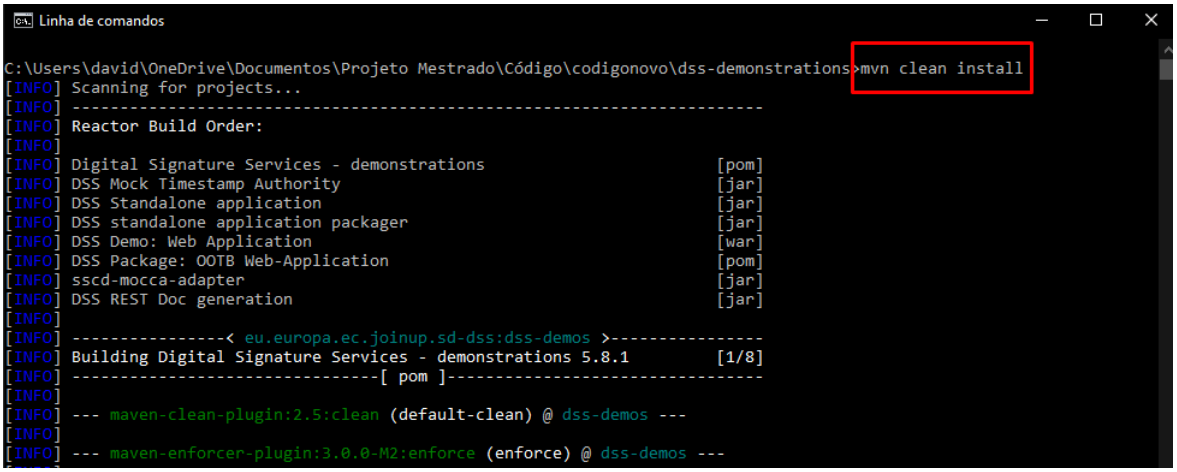


Figura 4.5: Comando utilizado para compilar o DSS Bundle.

Após o término da compilação do código, é possível notar as mensagens de sucesso, Figura 4.6.

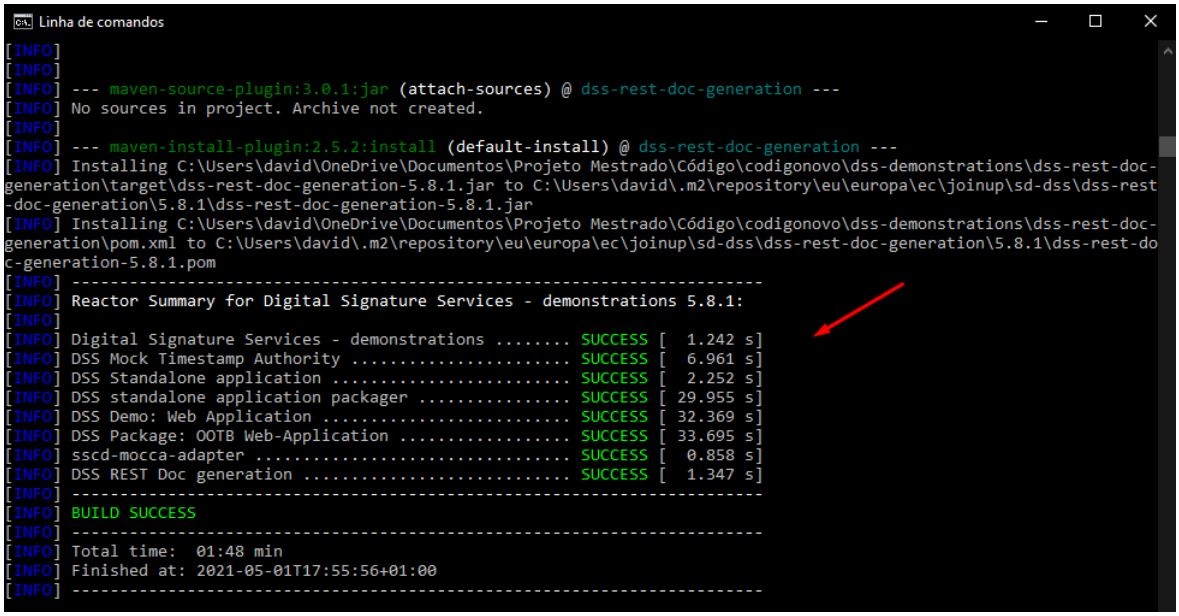


Figura 4.6: Sucesso na compilação do DSS Bundle.

Em seguida, é necessário a criação de um ficheiro `Dockerfile`, Bloco de Código 4.1, no qual contém as instruções a serem seguidas pelo Docker.

```
1 Base Alpine Linux based image with OpenJDK JRE only
2 FROM openjdk:15-jdk-alpine
3 copy application WAR (with libraries inside)
4 ADD ./dss-demo-bundle-5.8.RC1 /dss
5 specify default command
6 CMD "/dss/Webapp-Startup.sh"
```

Bloco de Código 4.1: Ficheiro Dockerfile.

Na linha 2 do `Dockerfile` do ficheiro acima, a instrução `FROM` especifica a imagem-pai a partir da qual será construída, neste caso, o Java JDK. Na linha 4, a instrução `ADD` copia novos ficheiros ou diretórios da origem e os adiciona ao sistema de ficheiro da imagem no caminho de destino. Neste caso, ele copia o ficheiro `WAR` do DSS *Bundle* e as bibliotecas e adiciona ao caminho `/dss`. Por fim, na linha 6, a instrução `CMD` executa o ficheiro `Webapp.Startup.sh` utilizado para iniciar o DSS *Bundle* e seus componentes.

O próximo passo é executar o Docker e configurar a porta que será utilizada para conexão com os restantes dos componentes do MVC. Para isto, é necessário rodar o comando `Build`, utilizado para construir imagens Docker a partir de um `Dockerfile` e de um "contexto". O contexto de uma construção é o conjunto de ficheiros localizados no `PATH` ou URL especificado.

A Figura 4.7 destaca a execução do comando `Build` para a criação da imagem Docker.

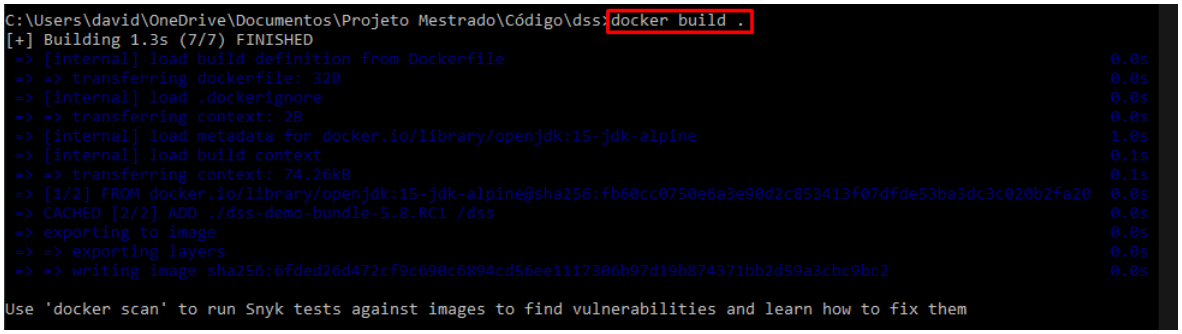


Figura 4.7: Execução do comando Build para construir a imagem do Docker.

Em seguida, é necessário executar o Docker especificando uma porta e o identificador da imagem, de acordo com a Figura 4.8. Podemos observar que o servidor Tomcat foi iniciado, isto indica que o DSS Bundle está no ar e pronto para receber as solicitações de validação do Controlador.

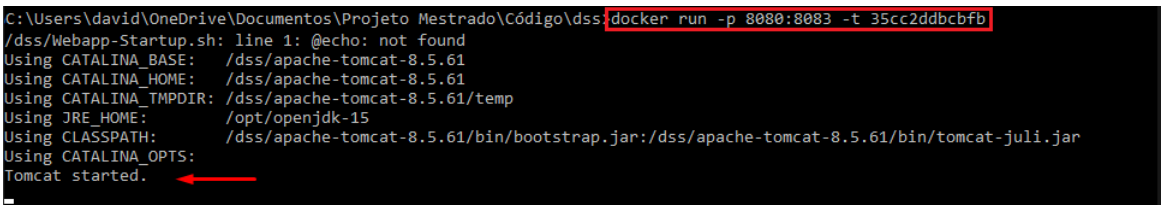


Figura 4.8: Execução do comando para iniciar a imagem do Docker.

Obs.: Depois de executar a imagem pela primeira vez, caso queira executar novamente, basta acionar o botão de *play* através do Docker *desktop*, Figura 4.9. Nota-se também a porta 8083 configurada previamente através da linha de comandos.

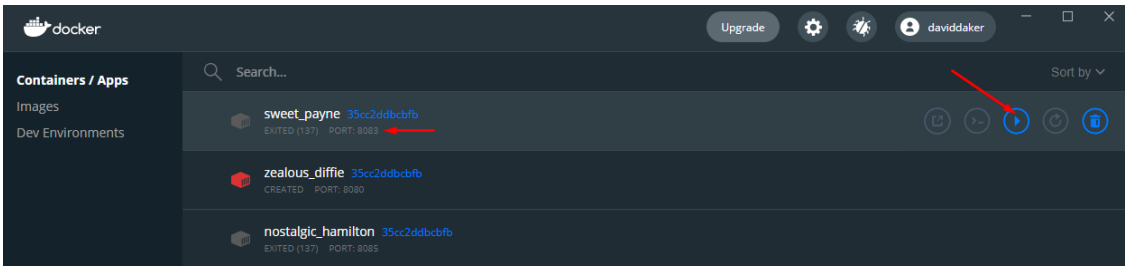


Figura 4.9: Botão para iniciar a imagem através do Docker desktop.

4.2.6 MVC - Controlador e Vista | REST API

Para a utilização do *Spring Boot* nesta aplicação, primeiramente foi necessário a criação do ficheiro `pom.xml` através do Spring Initializr (https://start.spring.io/), Figura 4.10. Este ficheiro contém as dependências necessárias para o projeto, tais como: o *Spring Web* para a REST API com o servidor Apache Tomcat e o Thymeleaf para a Linguagem de Marcação de Hipertexto -

HyperText Markup Language (HTML) ser carregado corretamente pelos navegadores, além de outras informações como nome do projeto, descrição, etc.

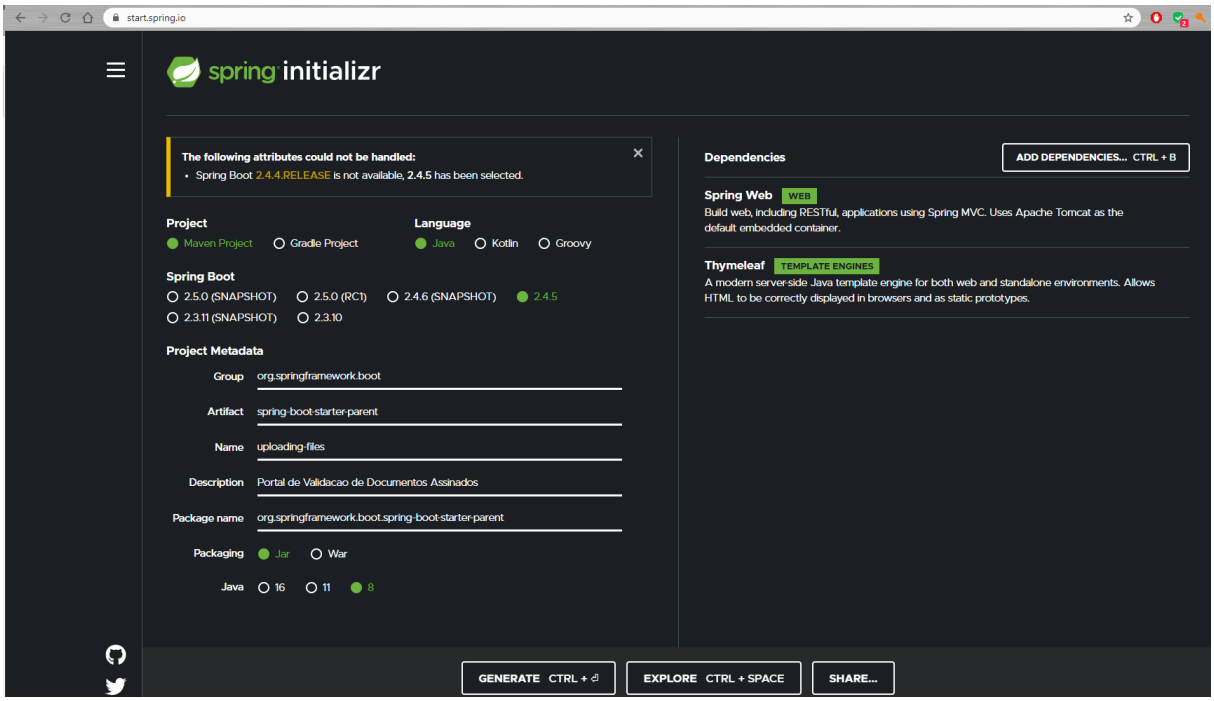


Figura 4.10: Criação do ficheiro pom.xml através do Spring Initializr.

```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
</dependency>
<dependency>
        <groupId>org.json</groupId>
        <artifactId>json</artifactId>
        <version>20180130</version>
</dependency>
```

Bloco de Código 4.2: Parte do ficheiro pom.xml.

Para a aplicação executar, é necessária uma classe `Starter` que literalmente vai iniciar a aplicação MVC, entretanto devido as auto-configurações do *Spring Boot*, isto já vem pronto, conforme o Bloco de Código 4.3 ilustra. Relativamente ao `@Bean`, sempre que a classe é inicializada pelo *Spring Boot*, o `CommandLineRunner` exclui e recria a pasta que contém os ficheiros enviados pelo utilizador.

```
@SpringBootApplication
@EnableConfigurationProperties(StorageProperties.class)
public class UploadingFilesApplication {

   public static void main(String[] args) {
      SpringApplication.run(UploadingFilesApplication.class, args);
   }

   @Bean
   CommandLineRunner init(StorageService storageService) {
      return (args) -> {
         storageService.deleteAll();
         storageService.init();
      };
   }
}
```

Bloco de Código 4.3: Classe para iniciar a aplicação MVC.

Já a classe `FileUploadController.java` é responsável pelo Controlador, ou seja, o

Spring MVC pega e direciona os pedidos para a rota correspondente. Por ser o Controlador, ela possui a anotação `@Controller`, conforme ilustra o Bloco de Código 4.4.

```
import java.nio.file.*;
@Controller
public class FileUploadController {
   private final StorageService storageService;
   @Autowired
   public FileUploadController(StorageService storageService) {
      this.storageService = storageService;}
```

Bloco de Código 4.4: Indicação do controlador.

Cada método possui uma *tag* chamada `@GetMapping` ou `@PostMapping` para conectar o caminho da rota com a ação HTTP correspondente do controlador.

O primeiro método, `@GetMapping(value = {"/myVar", "/"})`, é utilizado para buscar a lista de ficheiros que foram enviados pelos utilizadores para o serviço de armazenamento `(StorageService)`. Logo em seguida, carrega o resultado da validação no *template* do Thymeleaf, no caso, a página HTML. Para além disto, o componente `MvcUriComponentsBuilder` é responsável por gerar o *link* para o ficheiro a ser carregado. Neste caso, `myVar` corresponde à identificação do ficheiro, de acordo com o Bloco de Código 4.5.

```
@GetMapping(value = {"/{myVar}", "/"})
public String listUploadedFiles(Model model, @PathVariable(required = false)
    String myVar, @RequestParam(value="signatureDetails", required = false)
    List<String> attr, @RequestParam(value="indicationImage", required = false)
    String indicationImage) throws IOException {
      model.addAttribute("files", storageService.loadAll().map(
         path ->
             MvcUriComponentsBuilder.fromMethodName(FileUploadController.class,
               "serveFile",
                   path.getFileName().toString()).build().toUri().toString())
            .collect(Collectors.toList()));
        model.addAttribute("myVar", myVar);
        if (attr!=null)
          model.addAttribute("signatureDetails", attr);
        if (indicationImage!=null)
          model.addAttribute("indicationImage", indicationImage);
        else
          model.addAttribute("indicationImage", "white.png");
      return "uploadForm";
   }
```

Bloco de Código 4.5: Método para carregar o ficheiro enviado pelo utilizador no navegador.

A interface do serviço de armazenamento `StorageService` é ilustrada no Bloco de Código 4.6. Ela declara vários métodos abstratos para inicializar, armazenar, remover e recuperar

os ficheiros enviados pelo utilizador. Entretanto, ele lista apenas as operações de armazenamento possíveis, a implementação destes métodos é realizada em outra classe.

```
public interface StorageService {
   void init();
   void store(MultipartFile file);
   Stream<Path> loadAll();
   Path load(String filename);
   Resource loadAsResource(String filename);
   void deleteAll();}
```

Bloco de Código 4.6: Interface do serviço de armazenamento.

O próximo método, `@GetMapping("/export/type/{filename:.+}")`, é responsável por permitir ao utilizador realizar o *download* dos relatórios básico e avançado. Para isto, ele busca pelo nome do ficheiro dentro da pasta `files` localmente e realiza o *parse* do ficheiro JSON, caso o encontre. Como resposta, o utilizador recebe um ficheiro chamado `validations.txt`, de acordo com o Bloco de Código 4.7.

```
@GetMapping("/export/{type}/{filename:.+}")
public ResponseEntity<String> export(@PathVariable String filename,
    @PathVariable String type) {
    InputStream is = null;
    byte[] bytes = null;
    try {
    URL url = new URL("http://localhost:8080/files/"+filename);
        is = url.openStream ();
        bytes = IOUtils.toByteArray(is);
        if (is != null) is.close();
    } catch (IOException e) {
    }
    String validationResultJSON = null;
    validationResultJSON = getValidationResult(bytes);
    JSONObject obj = new JSONObject(validationResultJSON);
    String report = null;
    if (type.equals("simple") ) {
        report = obj.getJSONObject("SimpleReport").toString(4);
    } else {
        report = obj.getJSONObject("SimpleReport").toString(4) +
            obj.getJSONObject("DetailedReport").toString(4);
    }
    return ResponseEntity.ok().header("Content-Disposition","attachment;
    filename=validations.txt").header(HttpHeaders.CONTENT_TYPE,
        "text/plain").body(report);
}
```

Bloco de Código 4.7: Método que permite o download do relatório básico e avançado.

Para além disto, o primeiro botão contém um atributo `onclick="generatePDF()"` utilizado para invocar a função de criação do relatório básico em formato PDF. O segundo botão possui definido em seus atributos o *link* para a rota e a identificação do ficheiro através da variável `myVar`, que direciona para o *download* do relatório avançado, Bloco de Código 4.8.

```
<button id="download" onclick="generatePDF()" class="btn
    btn-outline-success">Relatorio Basico</button>
<a id="buttonAdvancedReport" class="btn btn-outline-danger"
    th:href="@{${'http://localhost:8080/export/detailed/' + myVar}}">Relatorio
    Avancado</a>
```

Bloco de Código 4.8: Atributos dos botões no HTML da página.

Ao selecionar no botão para *download* do relatório básico, a biblioteca `html2pdf` utiliza o id `results2` da estrutura HTML que contém o resultado da validação. A partir disto, é gerado o relatório em formato PDF com as opções definidas de acordo com a documentação da biblioteca, Bloco de Código 4.9.

```
<script>
function generatePDF() {
 var element = document.getElementById("results2");
 var opt = {
 margin:       1,
 filename:     'relatorioBasico.pdf',
 image:        { type: 'jpeg', quality: 1 },
 pagebreak: { mode: ['avoid-all', 'css', 'legacy'] },
 html2canvas:  { scale: 2 },
 jsPDF:        { unit: 'in', format: 'a4', orientation: 'portrait' }
 };
html2pdf(element,opt);
}
</script>
```

Bloco de Código 4.9: Script utilizado para criação do relatório básico em formato PDF.

A utilização da biblioteca `html2pdf` acontece ao carregar o *script* no cabeçalho da página HTML, conforme ilustra o Bloco de Código 4.10.

```
<script
    src="https://cdnjs.cloudflare.com/ajax/libs/html2pdf.js/0.8.0/html2pdf.bundle
    .min.js"></script>
```

Bloco de Código 4.10: Script para carregar biblioteca html2pdf.

O seguinte Bloco de Código 4.11, é responsável por realizar a conexão com o Modelo DSS Bundle, para que o ficheiro submetido pelo utilizador seja validado. Pode-se observar que a porta definida é a 8083, previamente configurada no Docker. Além disso, para que o ficheiro PDF possa ser enviado para validação, é necessário ele seja convertido para Base64 e adicionado no campo `bytes` na `String Payload`. Esta formatação do *payload* é definida na documentação do DSS *Bundle*. Como resultado, se obtém o ficheiro JSON contendo todas as validações realizadas.

```
private String getValidationResult(byte [] file) {
    String validationResultJSON="";
    try {
        String pdfBase64 = java.util.Base64.getEncoder().encodeToString(file);
        String payload =
            "{\"signedDocument\":{\"bytes\":\""+pdfBase64+"\",\"digestAlgorithm
            \":null,\"name\":\"pades-detached.xml\"},\"originalDocuments
            \":[{\"bytes\":\"\",\"digestAlgorithm\":null,\"name\":\"sample.xml
            \"}],
            \"policy\":null,\"tokenExtractionStrategy\":\"NONE\",\"signatureId
            \":null}";
        byte[] out =  payload.getBytes();
        int length = out.length;
        URL url = new URL
            ("http://localhost:8083/services/rest/validation/validateSignature");
        URLConnection con = url.openConnection();
        HttpURLConnection http = (HttpURLConnection)con;
        http.setRequestMethod("POST");
        http.setDoOutput(true);
        http.setFixedLengthStreamingMode(length);
        http.setRequestProperty("Content-Type", "application/json;
            charset=UTF-8");
        http.connect();
        try(OutputStream os = http.getOutputStream()) {
              os.write(out);
              os.flush();
              os.close();
        }
        BufferedReader reader = new BufferedReader(new
            InputStreamReader(con.getInputStream()));
        String line="";
        while ((line = reader.readLine()) != null) {
           validationResultJSON = validationResultJSON + line;
        }
        } catch(IOException e) {
        }
        return validationResultJSON;
}
```

Bloco de Código 4.11: Código responsável por enviar o ficheiro PDF para validação.

O ficheiro responsável pelas propriedades da aplicação é o `application.properties`, Bloco de Código 4.12. Ele está configurado para limitar o tamanho dos ficheiros submetidos pelos utilizadores a 100MB, linha 1. Na linha 2, está configurado para que tamanho total da solicitação para um `multipart/form-data` não exceda 100MB. E, por fim, na linha 3, não permite que a página de erro *default* (*Whitelabel Error Page*) do *Spring Boot* seja carregada para o utilizador, pois a página personalizada será exibida.

```
1 spring.servlet.multipart.max-file-size=100MB
2 spring.servlet.multipart.max-request-size=100MB
3 server.error.whitelabel.enabled=false
```

Bloco de Código 4.12: Ficheiro de configuração da aplicação.

Caso o ficheiro tenha algum problema durante o armazenamento, o Bloco de Código 4.13 é responsável por retornar a resposta 404 *Not Found* e consequentemente a tela de erro personalizada da Figura 4.11.

```
@ExceptionHandler(StorageFileNotFoundException.class)
   public ResponseEntity<?>
       handleStorageFileNotFound(StorageFileNotFoundException exc) {
        return ResponseEntity.notFound().build();
   }
```

Bloco de Código 4.13: Código para tratar exceções dos ficheiros.

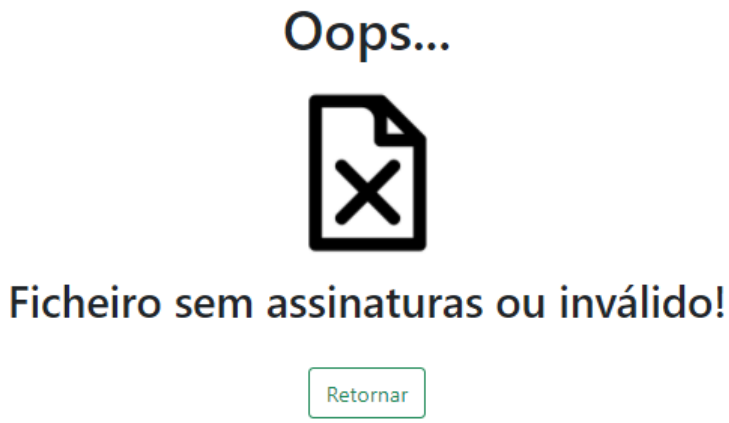


Figura 4.11: Tela de erro personalizada.

O seguinte Bloco de Código (4.14) realiza a primeira parte da busca de informações no ficheiro JSON vindo do *Bundle DSS*. Ele é responsável por extrair informações como: Nome do Documento, Política de Validação, a Data e Hora de Validação do Documento, Quantidade de Assinaturas e Quantidade de Assinaturas Válidas, e, em seguida, apresentar no HTML.

```
@PostMapping("/")
    public String handleFileUpload(@RequestParam("file") MultipartFile file,
         RedirectAttributes redirectAttributes) {
      storageService.store(file);
(...)

String documentName =
    obj.getJSONObject("SimpleReport").getString("DocumentName");
String policyName = obj.getJSONObject("SimpleReport").getJSONObject
    ("ValidationPolicy").getString("PolicyName");
String validationDate =
    obj.getJSONObject("DiagnosticData").getString("ValidationDate");
Integer signaturesCount =
    obj.getJSONObject("SimpleReport").getInt("SignaturesCount");
Integer validSignaturesCount =
    obj.getJSONObject("SimpleReport").getInt("ValidSignaturesCount");
(...)

headerDocument += "<hr><b>Nome do Documento: </b>" + file.getOriginalFilename();
headerDocument += "<b><br/>Politica de Validacao: </b>" + policyName;
//Get System Date and Time
DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH:mm:ss");
LocalDateTime now = LocalDateTime.now();
headerDocument += "<b><br/>Data e Hora da Validacao: </b>" + dtf.format(now);
headerDocument += "<b><br/>Quantidade de Assinaturas: </b>" +
    signaturesCount.toString();
headerDocument += "<b><br/>Quantidade de Assinaturas Validas: </b>" +
    validSignaturesCount.toString();
(...)
```

Bloco de Código 4.14: Parte do código para realizar a busca das informações.

O Bloco de Código 4.15 é responsável por aceder ao ficheiro JSON resultante da validação realizada pelo DSS Bundle, e para cada assinatura realizar a classificação em Válida, Inválida ou Indeterminada com sua respectiva indicação visual de texto e ícone. Em seguida, continua a executar o *parse* no JSON num ciclo para obtenção do nome da pessoa que assinou, data e hora da assinatura, tipo de assinatura e os erros presentes na assinatura/certificado. Por fim, carrega no HTML a concatenação do resultado do *parse* de cada assinatura.

```
private List < String > createSimpleReport(JSONArray arrJson, String
    headerDocument) {
List < String > signatures = new ArrayList < > ();
signatures.add(headerDocument);
String warnings\_str = "</br><b>Erros: </b>";
for(int i = 0; i < arrJson.length(); i++) {
    String warnings_str_line = "";
    String indicationImage = "";
    String indication =
   arrJson.getJSONObject(i).getJSONObject("Signature").getString("Indication");
        if (indication.equals("TOTAL\_PASSED")) {
           indicationImage = "<h3><img src=\"/images/passicon.png\"
               width=\"25\" height=\"25\"/> Assinatura Valida</h3>";
        } else if (indication.equals("INDETERMINATE")) {
           indicationImage = "<h3><img src=\"/images/undeterminated.png\"
               width=\"25\" height=\"25\"/> Assinatura Indeterminada</h3>";
        } else {
           indicationImage = "<h3><img src=\"/images/notpassed.png\"
               width=\"25\" height=\"25\"/> Assinatura Invalida</h3>";
        }
    String signedBy = "<b>Assinado por: </b>" +
arrJson.getJSONObject(i).getJSONObject("Signature").getString("SignedBy");
    String signatureLevel = "<b><br/>Tipo de Assinatura: </b>" +
arrJson.getJSONObject(i).getJSONObject("Signature").getJSONObject
    ("SignatureLevel").getString("description");
    String signedTime = "<b><br/>Data e Hora da Assinatura: </b>" +
arrJson.getJSONObject(i).getJSONObject("Signature").getString("SigningTime");
    JSONArray errors =
arrJson.getJSONObject(i).getJSONObject("Signature").getJSONArray("Errors");
    for(int j = 0; j < errors.length(); j++) {
        String replaceString =
            errors.getString(j).replace(",","<span>&44;</span>");
        warnings\_str\_line += "<br/> - " + replaceString;}
    signatures.add(indicationImage +
        signedBy+signatureLevel+signedTime+warnings\_str+warnings\_str\_line);}
    return signatures;
```

Bloco de Código 4.15: Código responsável pelo resultado individual de cada assinatura do ficheiro.

O Bloco de Código 4.16 é responsável por realizar a avaliação Global com base nos resultados de cada assinaturas. Para cada assinatura, o código obtém a indicação do resultado da validação no JSON e retorna verdadeiro ou falso.

```
private boolean allIndeterminates(JSONArray arrJson, Integer signaturesCount) {
  for (int i = 0; i< signaturesCount; i++) {
    if (!arrJson.getJSONObject(i).getJSONObject("Signature").getString("Indicat
        ion").equals("INDETERMINATE")) {
        return false;}}
        return true;}

private boolean allValid(JSONArray arrJson, Integer signaturesCount) {
  for (int i = 0; i< signaturesCount; i++) {
    if (!arrJson.getJSONObject(i).getJSONObject("Signature").getString("Indicat
        ion").equals("TOTAL_PASSED")) {
        return false;}}
        return true;}

private boolean allInvalid(JSONArray arrJson, Integer signaturesCount) {
  for (int i = 0; i< signaturesCount; i++) {
    if (!arrJson.getJSONObject(i).getJSONObject("Signature").getString("Indicat
        ion").equals("TOTAL_FAILED")) {
        return false;}}
        return true;}

private boolean notInvalid(JSONArray arrJson, Integer signaturesCount) {
  for (int i = 0; i< signaturesCount; i++) {
    if (arrJson.getJSONObject(i).getJSONObject("Signature").getString("Indicat
        ion").equals("TOTAL_FAILED")) {
        return false;}}
        return true;}

private boolean atLeastOneInvalid(JSONArray arrJson, Integer signaturesCount) {
  for (int i = 0; i< signaturesCount; i++) {
    if (arrJson.getJSONObject(i).getJSONObject("Signature").getString("Indicat
        ion").equals("TOTAL_FAILED")) {
        return true;}}
        return false;}

private boolean atLeastOneIndeterminate(JSONArray arrJson, Integer
    signaturesCount) {
  for (int i = 0; i< signaturesCount; i++) {
    if (arrJson.getJSONObject(i).getJSONObject("Signature").getString("Indicat
        ion").equals("INDETERMINATE")) {
        return true;}}
        return false;}
```

Bloco de Código 4.16: Código para carregar o PDF no HTML.

De acordo com o resultado das condições acima, o Bloco de Código 4.17 carrega o respectivo resultado (Válida, Inválida ou Indeterminada) no HTML. Para o resultado ser válido, todas assinaturas devem ser válidas. Para o resultado ser indeterminado, basta uma assinatura ser indeterminada. Por fim, para o resultado ser inválido, a assinatura deve ser inválida. Caso nenhuma das condições se verifiquem, as assinaturas são classificadas como indeterminadas.

```
JSONArray jsonArray =
    obj.getJSONObject("SimpleReport").getJSONArray("signatureOrTimestamp");
    if (allValid(jsonArray, signaturesCount)) {
        indicationImage = "passicon.png";
        indicationResult = "Assinatura Valida";
        headerDocument += "<h1 style=\"color:green; text-align:
            center;\">Assinatura(as) Valida(as)</h1>\n";
    } else if (notInvalid(jsonArray, signaturesCount) &&
        atLeastOneIndeterminate(jsonArray, signaturesCount)) {
        indicationImage = "undeterminated.png";
        indicationResult = "Assinatura Indeterminada";
        headerDocument += "<h1 style=\"color:orange; text-align:
            center;\">Assinatura(as) Indeterminada(as)</h1>";
    } else if (atLeastOneInvalid(jsonArray, signaturesCount)) {
       indicationImage = "notpassed.png";
       indicationResult = "Assinatura Invalida";
       headerDocument += "<h1 style=\"color:red; text-align:
           center;\">Assinatura(as) Invalida(as)</h1>";
    } else {
       indicationImage = "undeterminated.png";
       indicationResult = "Assinatura Indeterminada";
       headerDocument += "<h1 style=\"color:orange; text-align:
           center;\">Assinatura(as) Indeterminada(as)</h1>";
    }
redirectAttributes.addAttribute("signatureDetails", createSimpleReport(arrJson,
    headerDocument) );
```

Bloco de Código 4.17: Parte do código para carregar resultado global das assinaturas.

O resultado da validação de cada assinatura (`signatureDetails`) e respectivo ícone (`indicationImage`) são carregados no HTML através do Bloco de Código 4.18.

```
<h1><img th:src="@{${'/images/'+indicationImage}}" width="70" height="70"
    alt=""/></h1>
<table id="tableIndication" class="table" scope="col2" aria-hidden="true">
    <tbody>
    <tr th:each="signatureDetail : ${signatureDetails}">
        <td valign="middle" th:utext="${signatureDetail}"></td></tr>
    </tbody>
</table>
```

Bloco de Código 4.18: Código para carregar o resultado das assinaturas no HTML.

Por fim, o Bloco de Código 4.19 é responsável por buscar o ficheiro PDF que será carregado no HTML. O código realiza um pedido *GET* utilizando o nome do ficheiro submetido.

```
@GetMapping("/files/{filename:.+}")
   @ResponseBody
   public ResponseEntity<Resource> serveFile(@PathVariable String filename) {
      Resource file = storageService.loadAsResource(filename);
      return ResponseEntity.ok().header(HttpHeaders.CONTENT_TYPE,
          "application/pdf").body(file);
   }
```

Bloco de Código 4.19: Código para carregar o PDF no HTML.

Em seguida, no HTML é invocada a variável `myVar` que contém o ficheiro e o retorna no corpo do HTML, Bloco de Código 4.20.

```
<embed id="pdf-js-viewer" th:src="@{${'http://localhost:8080/files/' + myVar}}"
    title="PDF"  frameborder="1" width="500" height="735">
```

Bloco de Código 4.20: Código para buscar o ficheiro PDF para ser carregado no HTML.

Capítulo 5

Experiências e Testes

5.1 Ambiente de Execução

Para a execução da aplicação, conforme mencionado no capítulo anterior, o Docker com o DSS *Bundle* deve estar a rodar, Figura 5.1. Conforme se pode notar, o Docker está a rodar na Porta 8083.

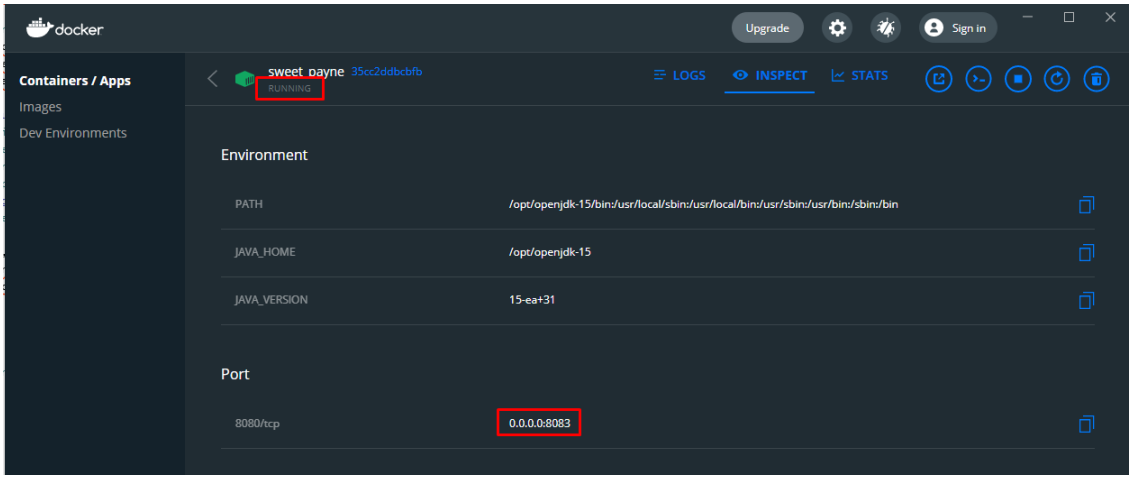


Figura 5.1: Docker com o DSS Bundle a rodar.

É importante ressaltar que a porta no código fonte deve ser a mesma porta do Docker e deve ser devidamente configurada antes da compilação e execução do *Spring boot*. O Bloco de Código 5.1 destaca o local onde a porta deve ser configurada.

```
URL url = new
    URL("http://localhost:8083/services/rest/validation/validateSignature");
```

Bloco de Código 5.1: Local no código fonte para configurar a porta.

Além disto, o *Spring Boot* deve estar a executar os seguintes componentes: Controlador da REST API e Visão, gerados no ficheiro `.JAR` através dos seguintes comandos:

```
cd C:\Users\david\..\Springboot\dss-main\complete
mvn clean install -Dmaven.test.skip=true
cd C:\Users\david\..\Springboot\dss-main\complete\target
java -jar uploading-files-0.0.1-SNAPSHOT.jar
```

Bloco de Código 5.2: Comandos para compilar e rodar o Spring Boot.

As Figuras 5.2 e 5.3 destacam, respectivamente, a compilação feita com sucesso e a página inicial da aplicação a funcionar no navegador.

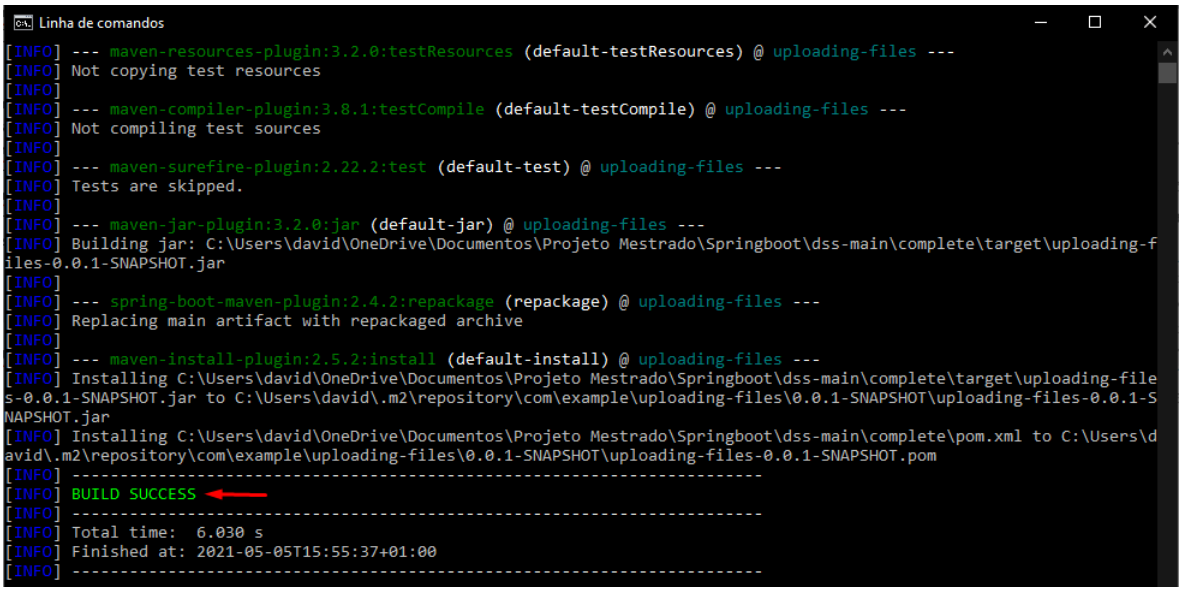


Figura 5.2: Compilação feita com sucesso.

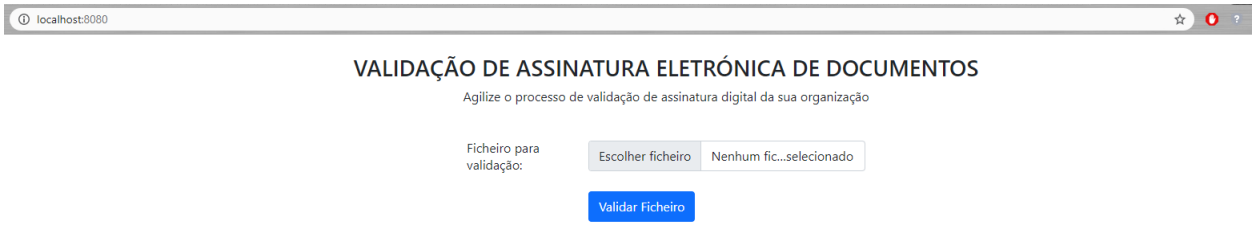


Figura 5.3: Página inicial da aplicação.

5.2 Processo de Utilização - Validação dos Ficheiros PDF

Após a configuração da aplicação, basta o utilizador selecionar “Escolher Ficheiro” e o botão “Validar Ficheiro”. **Observação:** O nome do ficheiro não deve conter caracteres especiais e espaços!

Caso 1: Assinatura presente no ficheiro PDF é inválida.

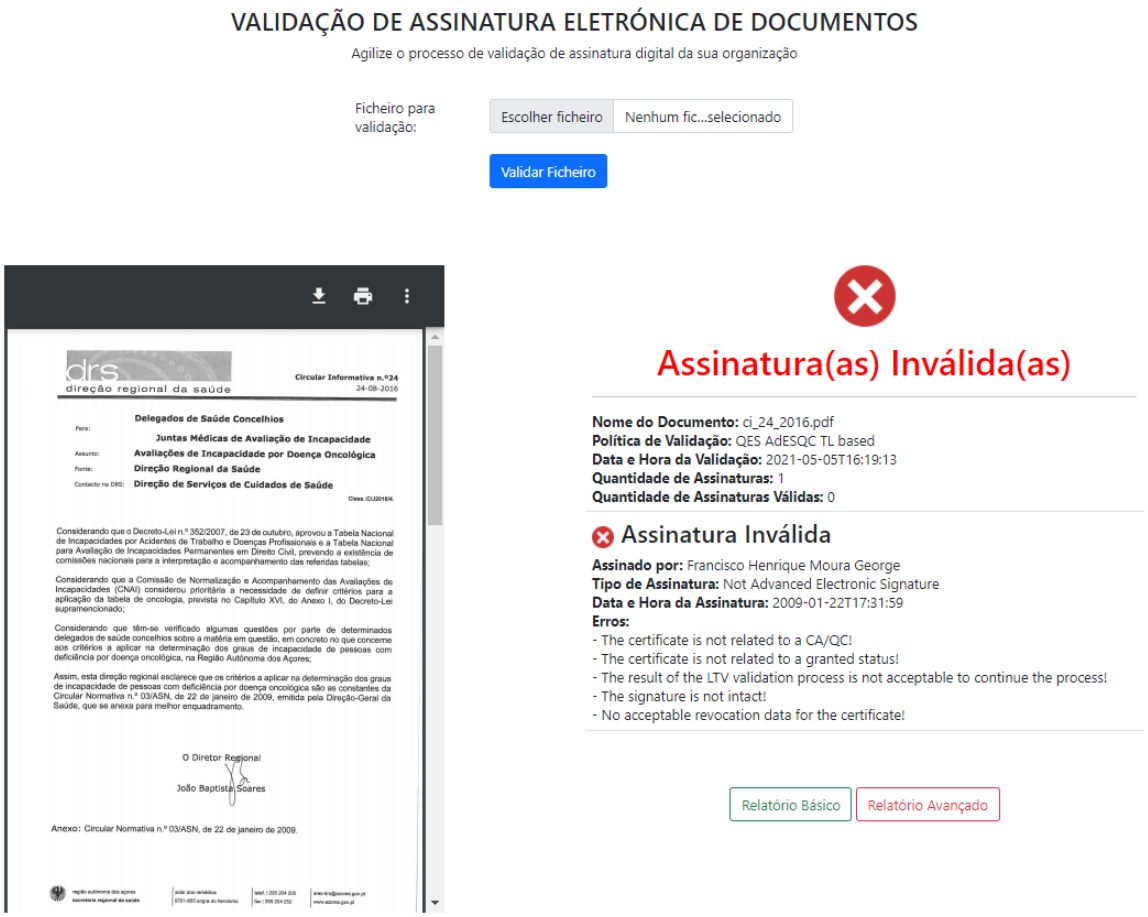


Figura 5.4: Assinatura inválida.

Para o caso acima, a validação da assinatura resultou inválida devido à vários erros, tendo como principais: A assinatura não está íntegra, nenhum dado de revogação aceitável para o certificado e o certificado não é relacionado à CA.

Caso 2: Assinatura presente no ficheiro PDF é indeterminada.

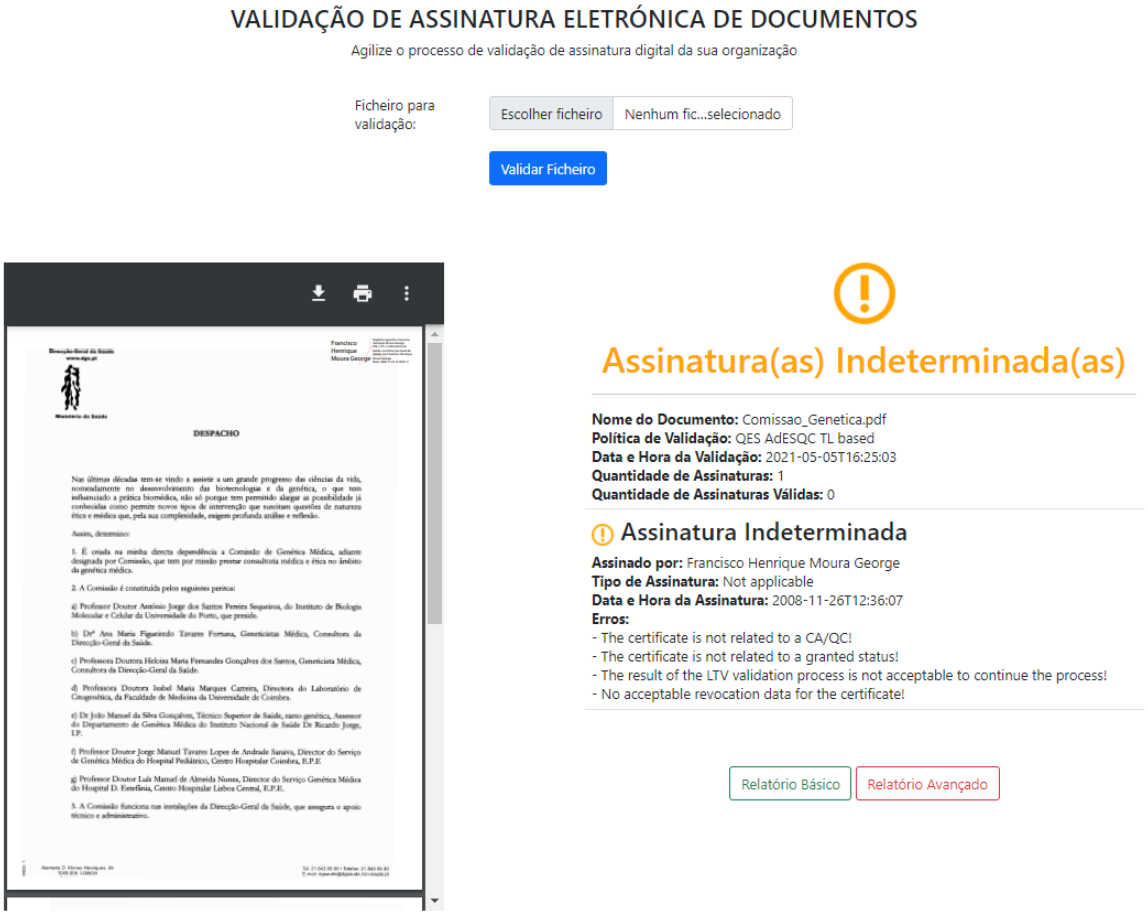
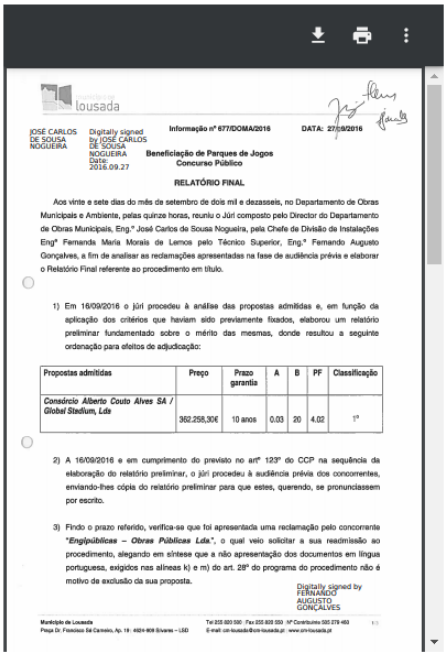


Figura 5.5: Assinatura indeterminada.

Para o caso acima, a validação da assinatura resultou indeterminada devido à vários erros, tendo como principais: o certificado não é relacionado à CA, nenhum dado de revogação aceitável para o certificado e o resultado do processo de Validação a Longo Prazo - Long Term Validation (LTV), não é aceitável para continuar o processo.



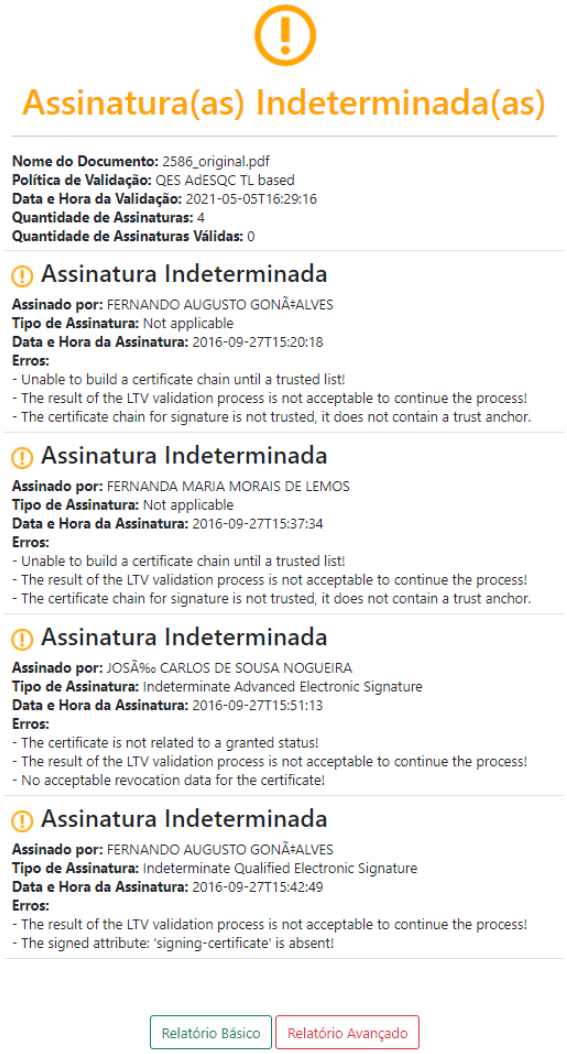


Figura 5.6: Mais de uma assinatura indeterminada.

Para o caso acima, a validação das assinaturas resultou indeterminada devido à vários erros, tendo como principais: não foi possível construir a cadeia de certificação até a lista confiável, o certificado não é relacionado à CA, a cadeia de certificados não é confiável, não contém uma âncora de confiança, nenhum dado de revogação aceitável para o certificado e o resultado do processo de validação do LTV não é aceitável para continuar o processo.

Caso 3: Misto de assinaturas indeterminadas e válidas presente no ficheiro PDF.

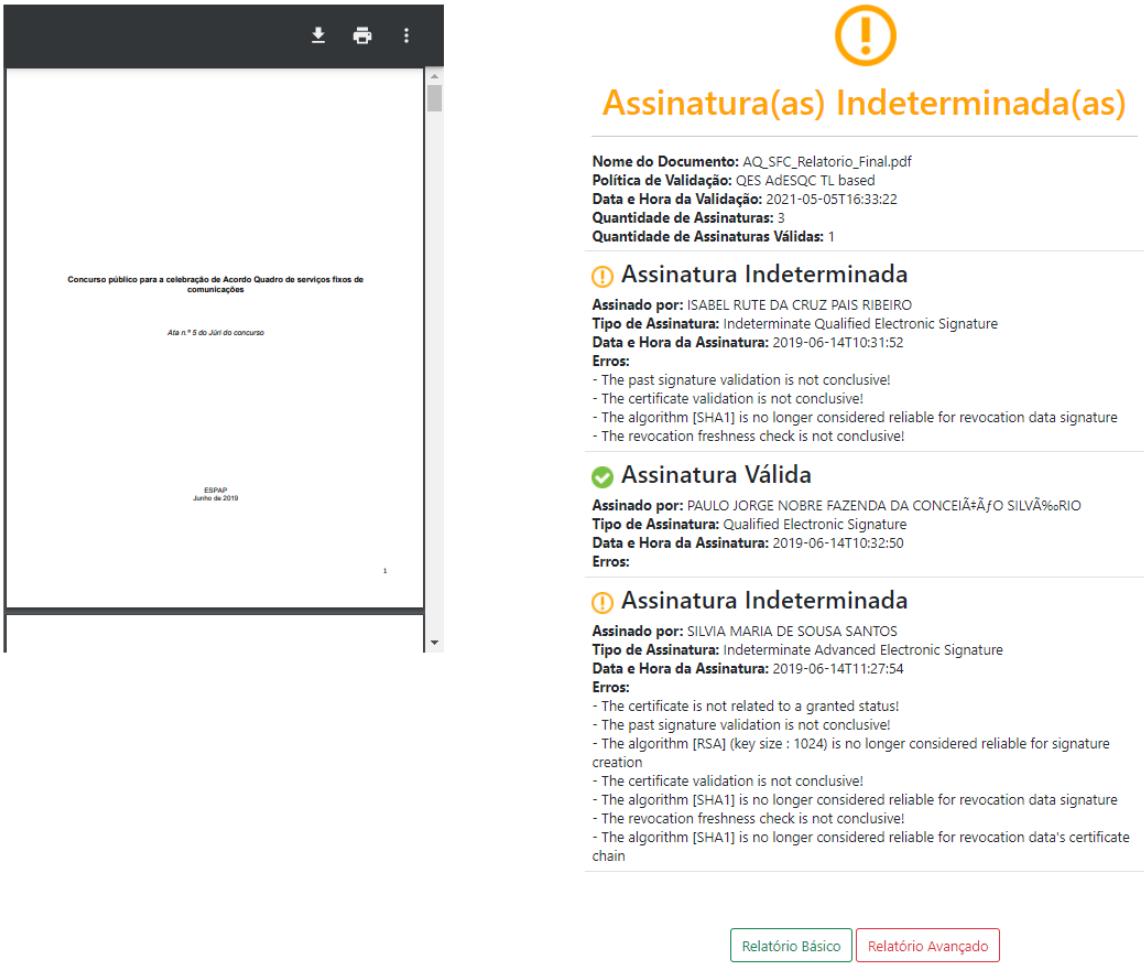


Figura 5.7: Misto de assinaturas indeterminadas e válidas.

Para o caso acima, apesar do resultado de uma das assinaturas ser válido, as outras são indeterminadas. De acordo com o que foi dito acima, basta uma assinatura ser indeterminada para que a avaliação global seja indeterminada.

Caso 4: Todas assinaturas do ficheiro PDF são válidas.

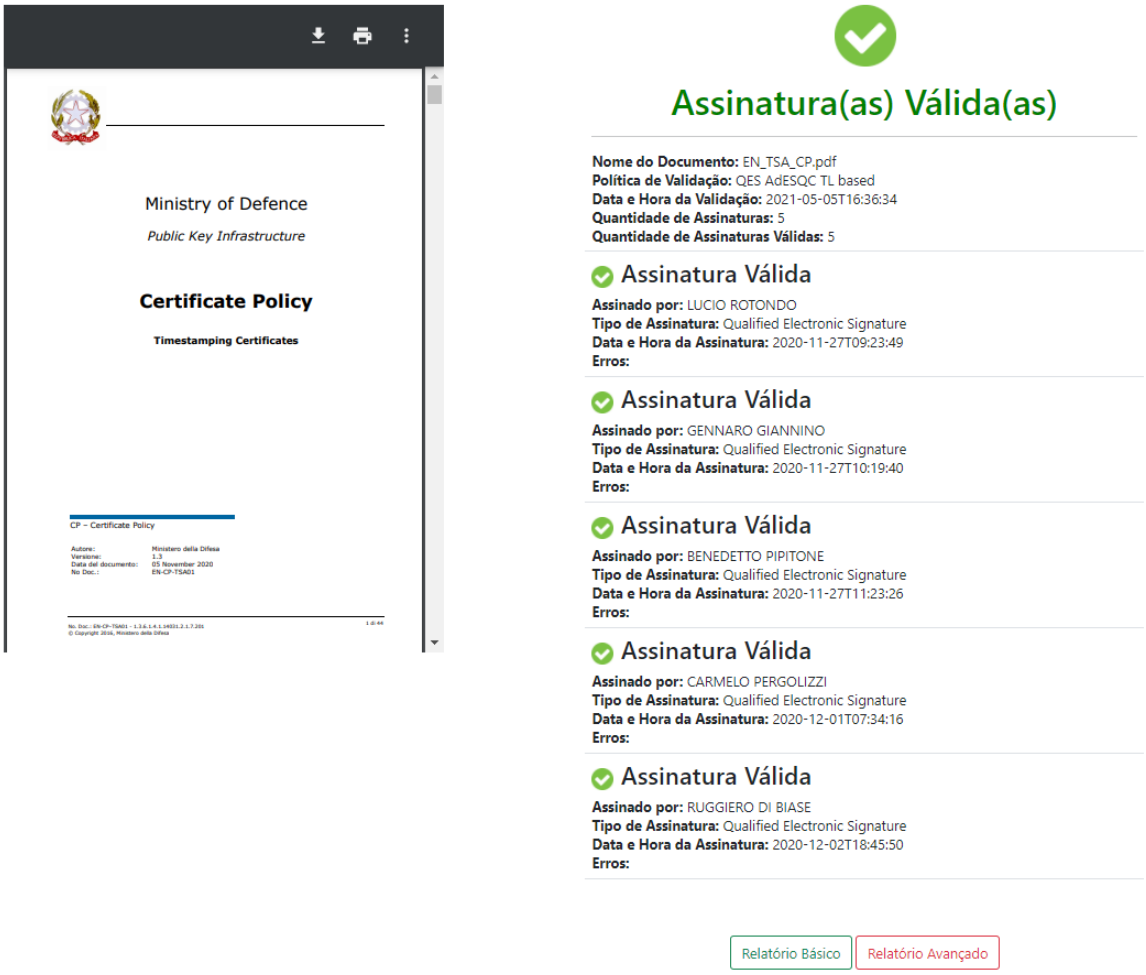


Figura 5.8: Assinaturas válidas.

Caso 5: Ficheiro inválido ou sem assinaturas.

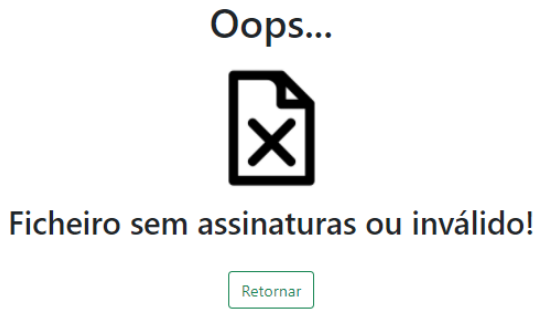


Figura 5.9: Ficheiro inválido ou sem assinaturas.

5.3 Processo de Utilização - Download dos Relatórios Básico e Avançado

É possível realizar o *download* do relatório básico e avançado do processo de validação realizado pelo DSS *Bundle*. Para isto, após a validação de um ficheiro PDF, basta selecionar o botão "Relatório Básico" ou "Relatório Avançado".

Figura 5.10: Download do relatório básico.

As Figuras 5.11 e 5.12 destacam parte do relatório básico e do relatório avançado após o *download* ser realizado.

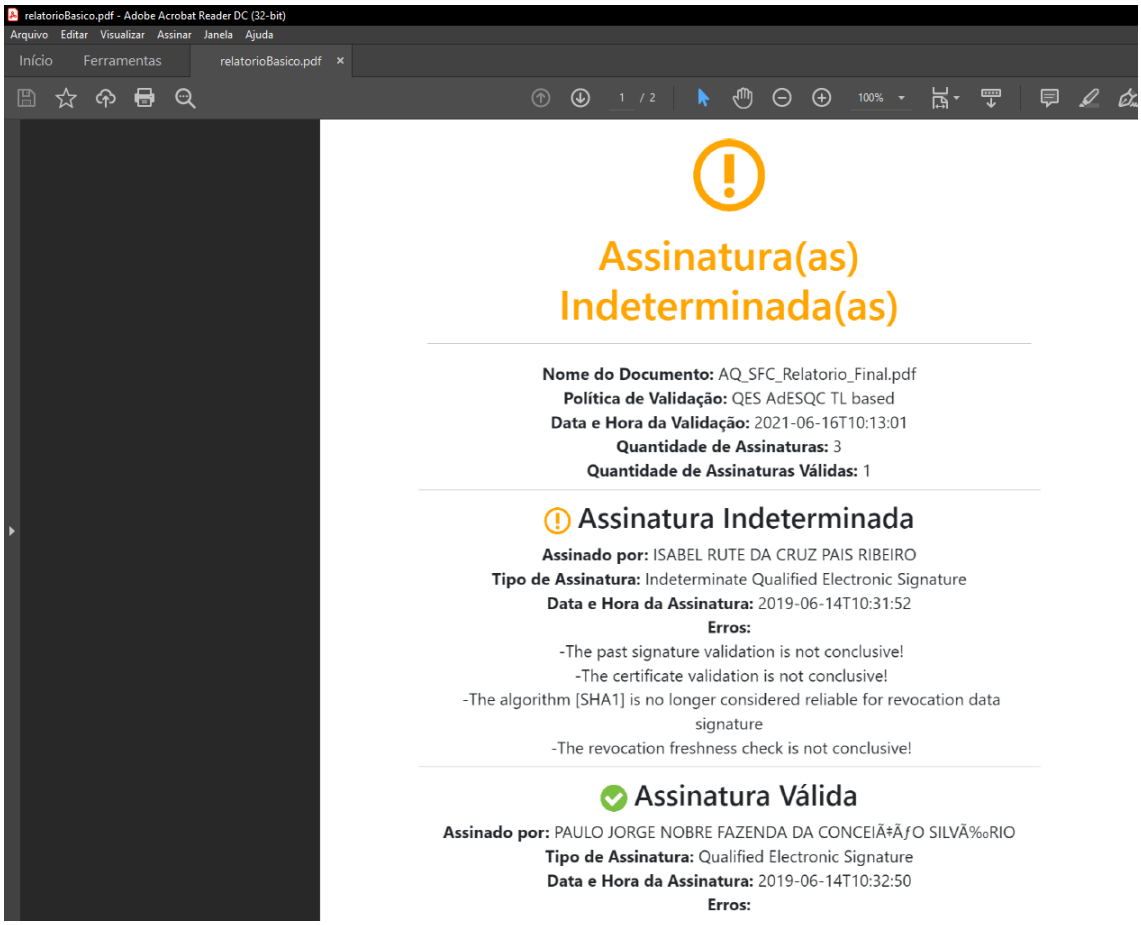


Figura 5.11: Parte do relatório básico.

validations (1).txt - Bloco de notas

Ficheiro Editar Formatar Ver Ajuda


```
"signatureOrTimestampOrCertificate": [
    {"Signature": {
        "ValidationProcessLongTermData": {
            "Constraint": [
                {
                    "Status": "OK",
                    "AdditionalInfo": null,
                    "Warning": null,
                    "Error": null,
                    "Id": null,
                    "Info": null,
                    "Name": {
                        "NameId": "LTV_ABSV",
                        "value": "Is the result of the Basic Validation Process acceptable?"
                    }
                },
                {
                    "Status": "OK",
                    "AdditionalInfo": null,
                    "Warning": null,
                    "Error": null,
                    "Id": "R-588B63966DD0ECDBC44677226505D48170AA58486B6F7525F2BA9DB29E545CDC",
                    "Info": null,
                    "Name": {
                        "NameId": "ADEST_RORPIIC",
                        "value": "Is the result of the revocation data validation process acceptable?"
                    }
                },
                {
```


Figura 5.12: Parte do relatório avançado.

5.4 Análise Estática de Código - Sonarqube

Para identificar potenciais vulnerabilidades e *bugs* no código, foi realizada a utilização do *software* Sonarqube. Observa-se na Figura 5.13 que foram encontrados 20 *bugs*, 0 vulnerabilidades e 3 pontos de melhorias de segurança.

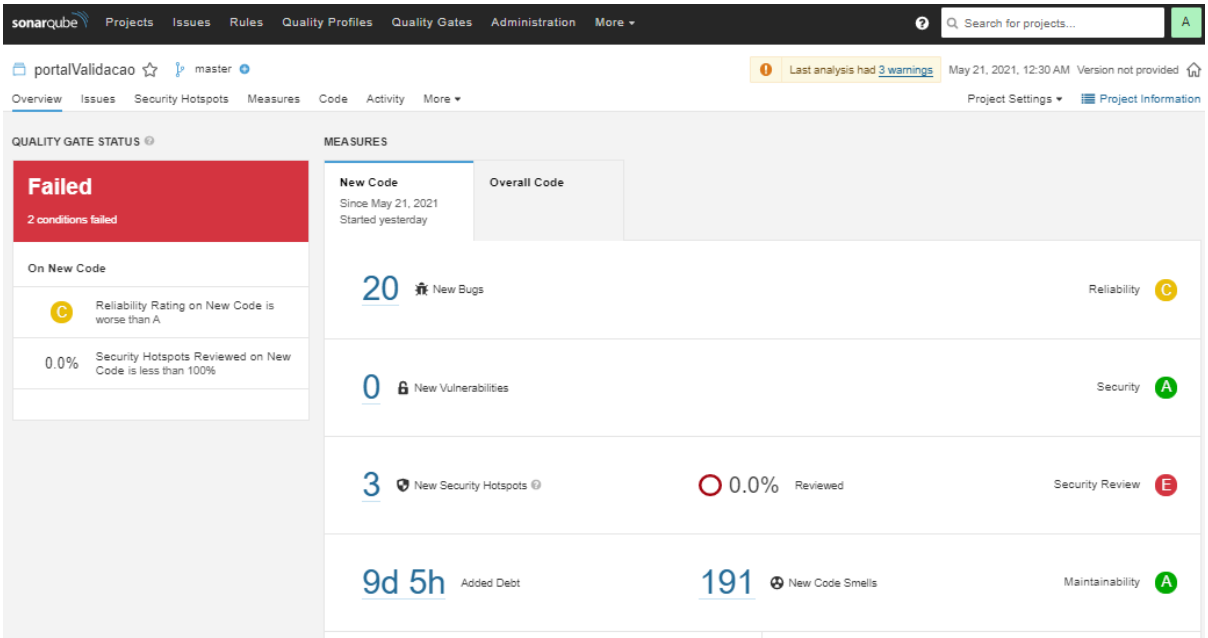


Figura 5.13: Avaliação geral do código.

Os 20 *bugs* são relacionados a biblioteca externa `bootstrap.css` utilizada no *template* do HTML, Figura 5.14.

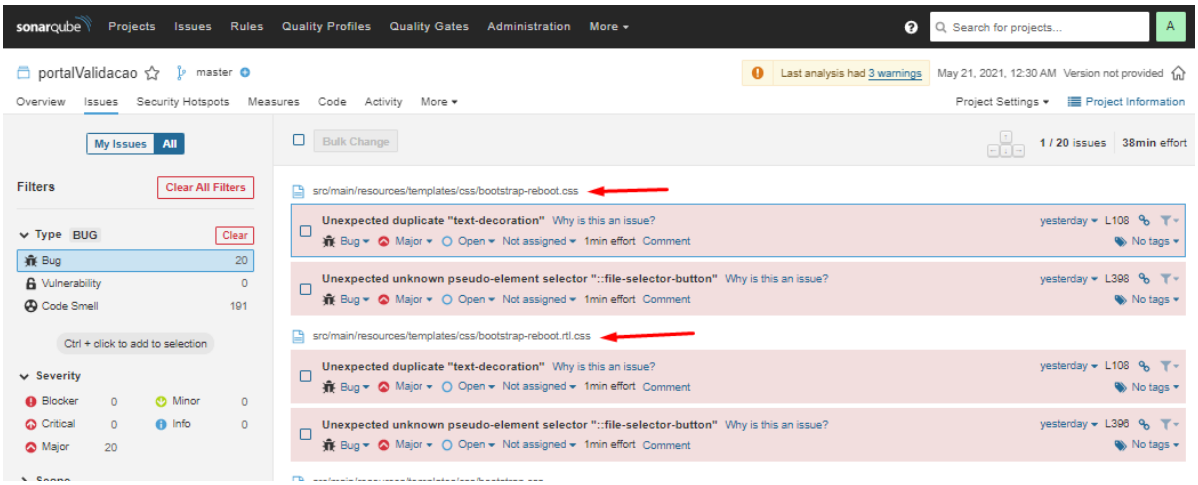


Figura 5.14: Parte dos bugs encontrados no bootstrap.

Os pontos de melhorias de segurança também estão relacionados com a biblioteca externa `bootstrap.css`, Figura 5.15.

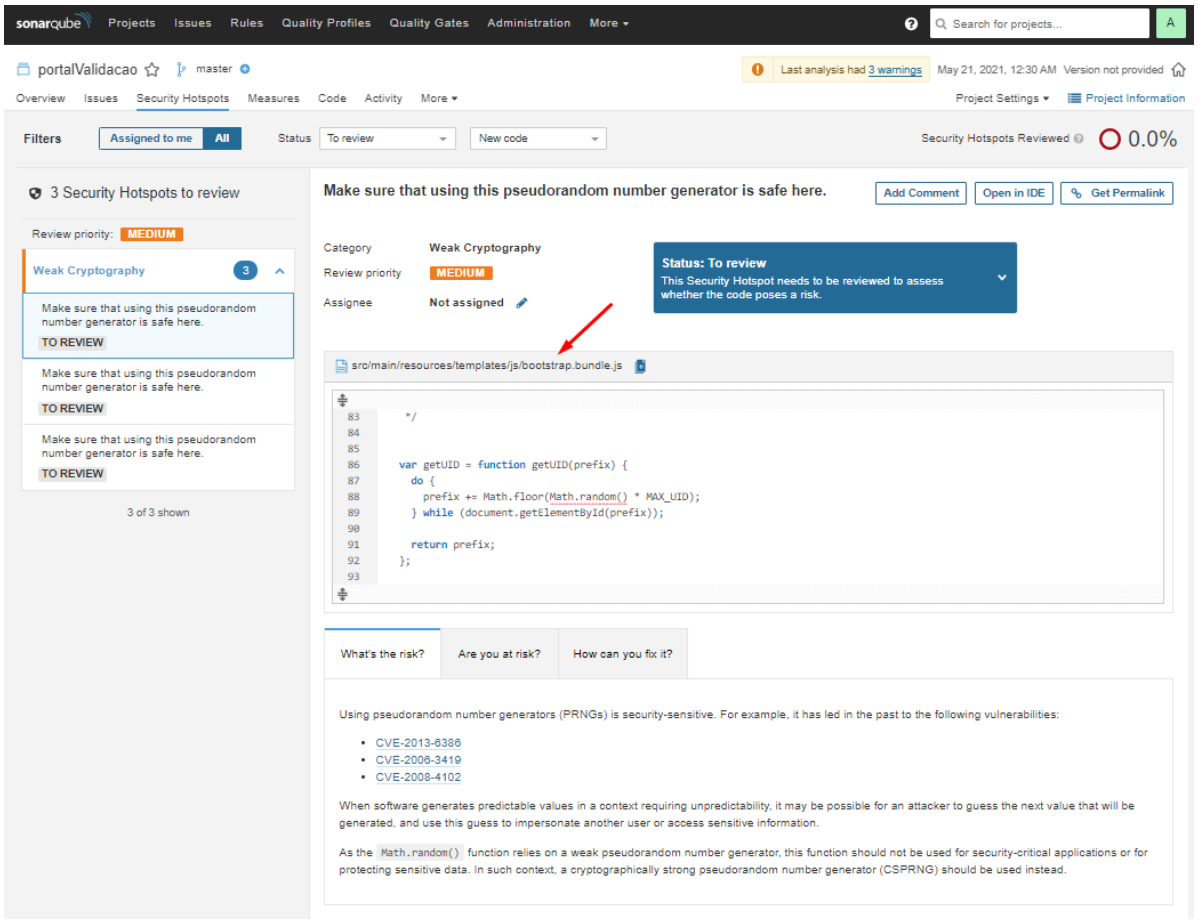


Figura 5.15: Pontos de melhoria de segurança encontrados no bootstrap.

Desta forma, por haverem estes pontos de segurança na biblioteca externa e alguns *bugs*, o Sonarqube classificou o código como *Failed*. Após discussão com a Loqr, estes pontos foram

considerados como falso-positivos e configurados como tal no Sonarqube. Sendo assim, o código recebeu a indicação de “Passed”, Figura 5.16.

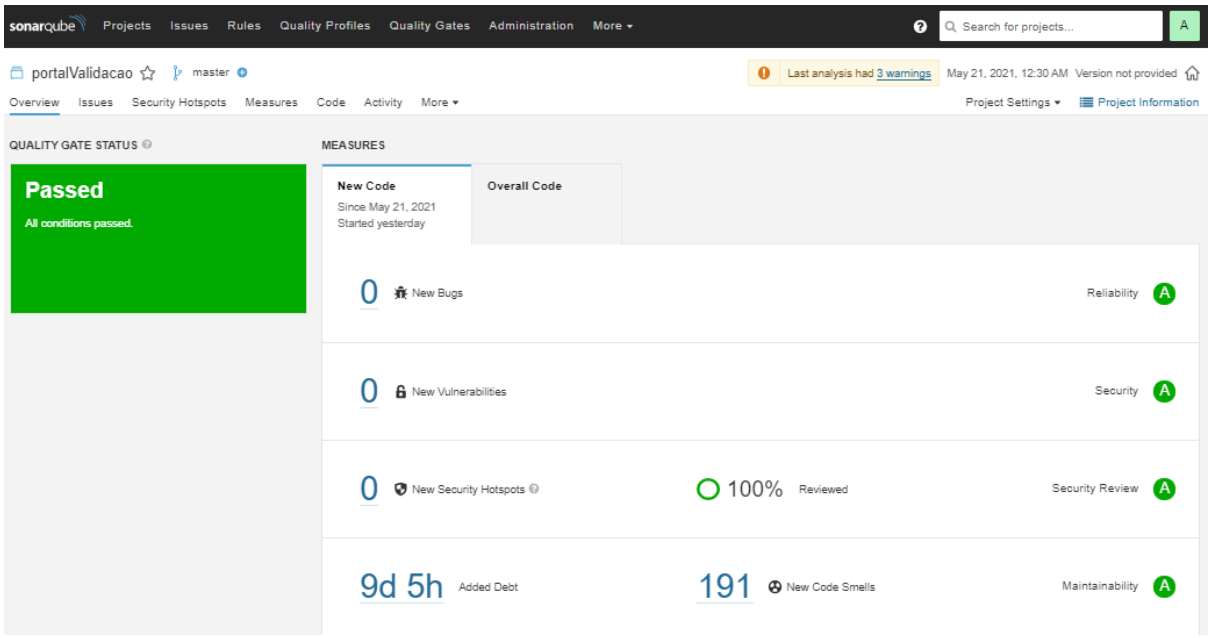


Figura 5.16: Resultado após classificação como falso-positivos.

Capítulo 6

Resultados e análise

Através das experiências e dos vários testes realizados no capítulo anterior, é possível afirmar que a arquitetura do sistema desenvolvido cumpre os objetivos definidos no início deste projeto, ou seja, o sistema é totalmente funcional e capaz de realizar as validações de forma eficaz. São realizadas as validações das assinaturas (tanto individualmente, como globalmente), a visualização do PDF submetido, além de permitir o *download* dos relatórios básicos e avançados contendo as validações realizadas pelo *DSS Bundle*.

É possível alegar que esta aplicação garante aos utilizadores uma interface intuitiva de utilização, além de apresentar de forma simples, rápida e de fácil percepção os resultados finais da validação. A Loqr ao utilizar esta aplicação, pode ser considerada como uma das empresas Portuguesas pioneiras na apresentação dos resultados do *DSS Bundle* de forma acessível e eficaz para todos os tipos de utilizadores. A Figura 6.1 destaca a validação de um ficheiro em formato PDF.

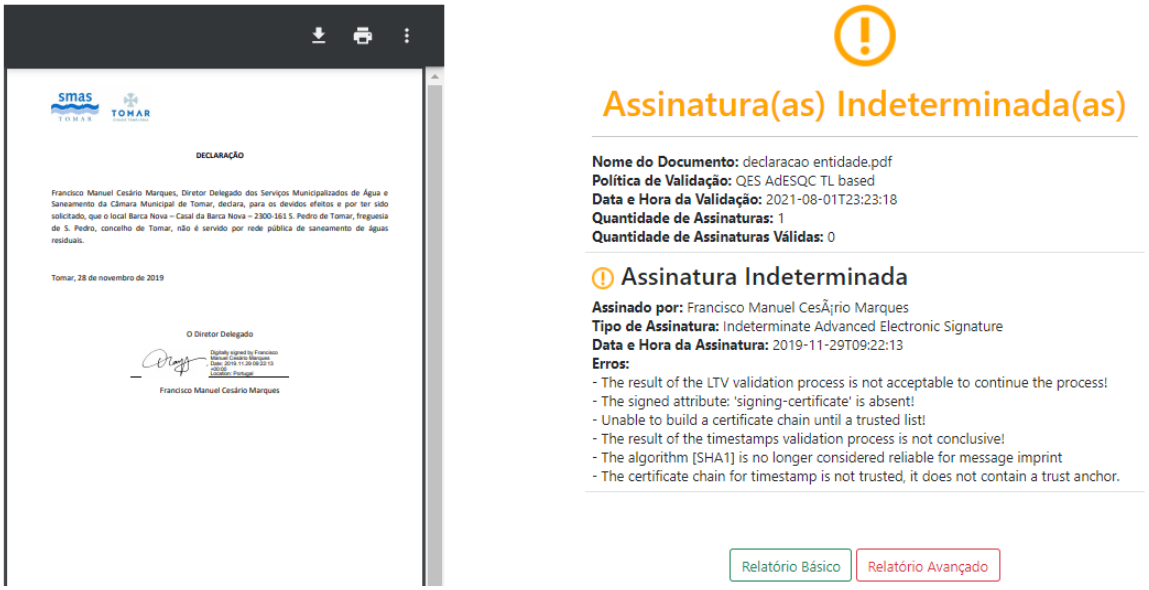


Figura 6.1: Validação de um ficheiro em formato PDF.

6.1 Performance

Relativamente à perfomance da aplicação, através dos casos de teste apresentados no capítulo anterior, pode-se inferir que:

- **Tempo de envio e validação:** O tempo de envio do ficheiro e da validação varia de acordo com o tamanho do ficheiro PDF, da quantidade de assinaturas e da velocidade da Internet do utilizador.

A Figura 6.2 destaca o tempo de envio e validação de um ficheiro PDF de `269KB` e 4 assinaturas indeterminadas.

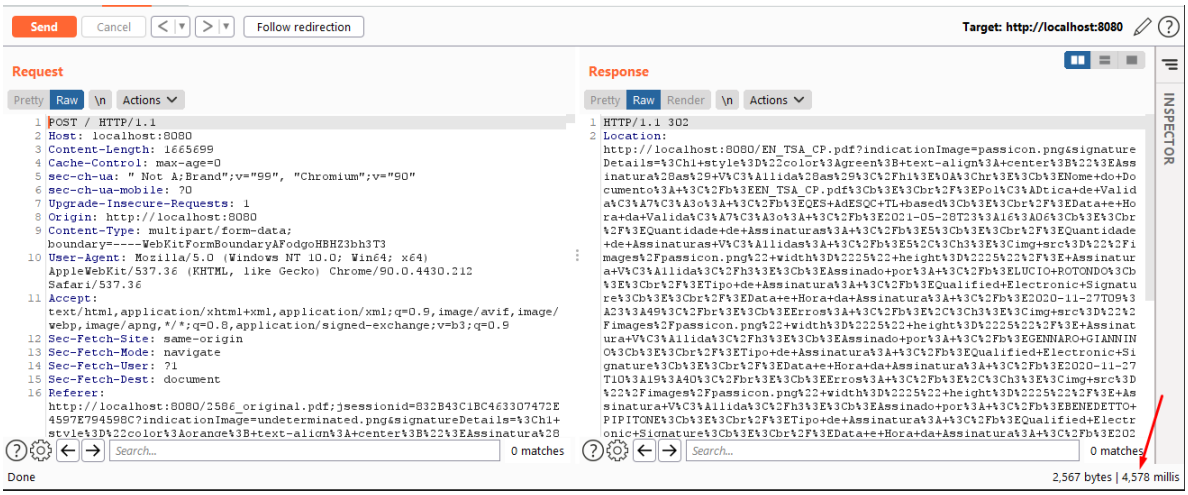


Figura 6.2: Tempo de resposta da validação de um ficheiro de 269KB.

A Figura 6.3 destaca o tempo de envio e validação de um ficheiro PDF de `914KB`, 2 assinaturas indeterminadas e 1 válida.

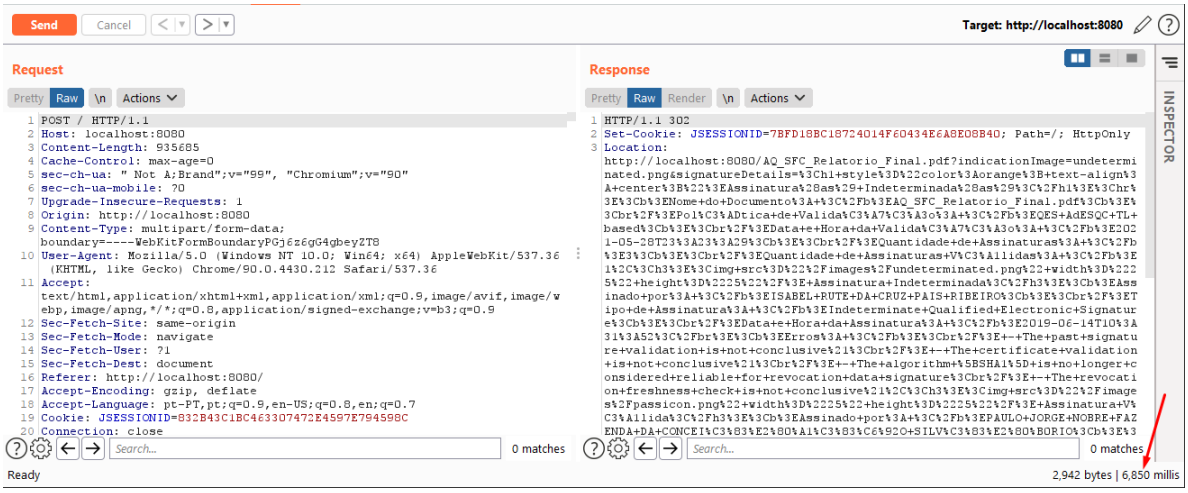


Figura 6.3: Tempo de resposta da validação de um ficheiro de 914KB.

A Figura 6.4 destaca o tempo de envio e validação de um ficheiro PDF de 2537KB e 1 assinatura indeterminada.

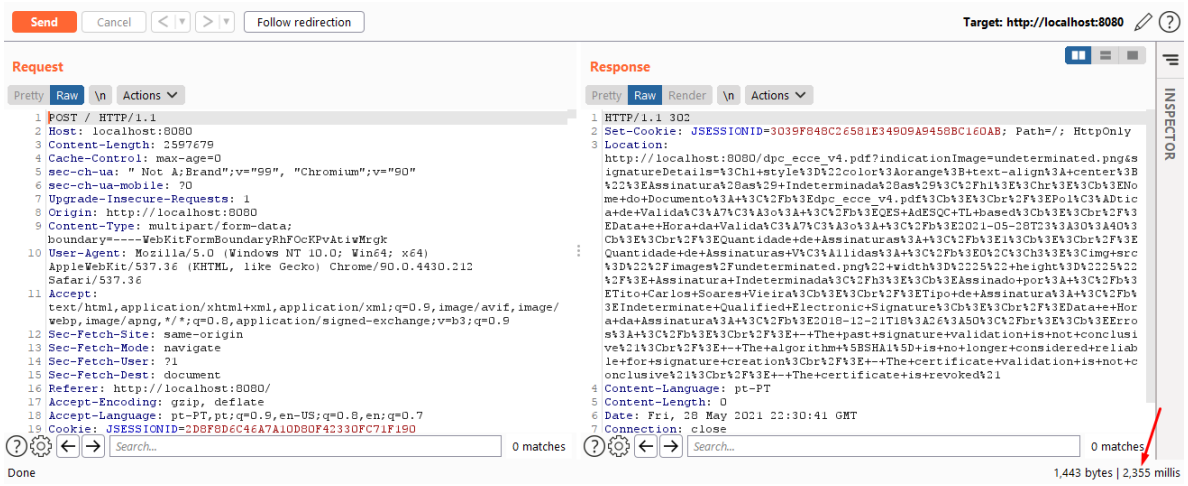


Figura 6.4: Tempo de resposta da validação de um ficheiro de 2537KB.

Por fim, a Figura 6.5 destaca o tempo de envio e validação de um ficheiro PDF de 1627KB e 5 assinaturas válidas.

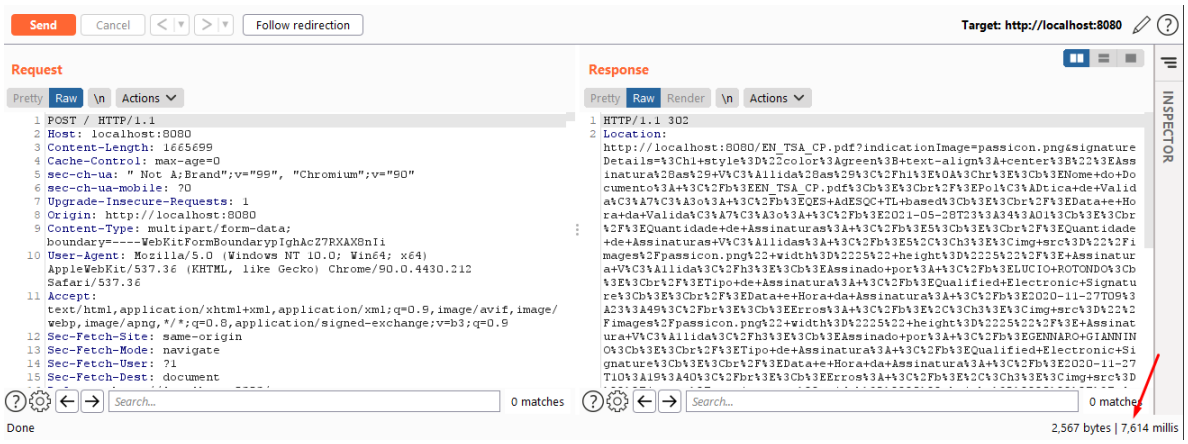


Figura 6.5: Tempo de resposta da validação de um ficheiro de 1627KB.

- **Tempo de carregamento da aplicação:** O portal está ser apresentado para o utilizador entre 4-6 milissegundos, salvo quando problemas de Internet, Figura 6.6.

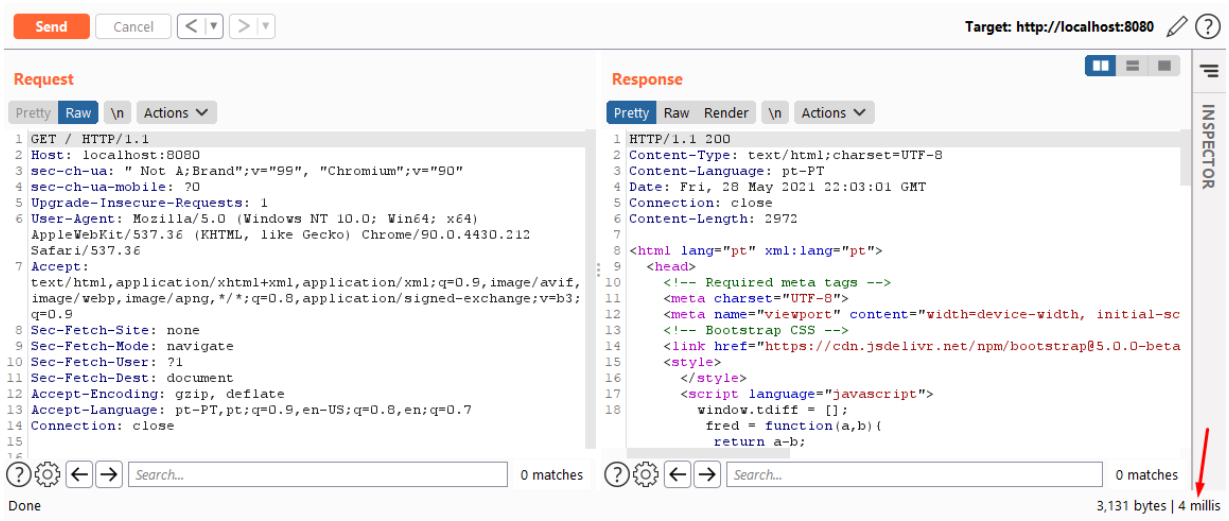


Figura 6.6: Aplicação a ser carregada em 4 milissegundos.

- **Tempo de criação do ficheiro com relatório básico ou avançado:** O tempo de criação do ficheiro com a análise básica ou avançada varia de acordo com a quantidade de assinaturas no ficheiro PDF.

Nas Figuras 6.7 e 6.8 é destacado o tempo de geração do relatório básico e avançado de um ficheiro com 5 assinaturas válidas.

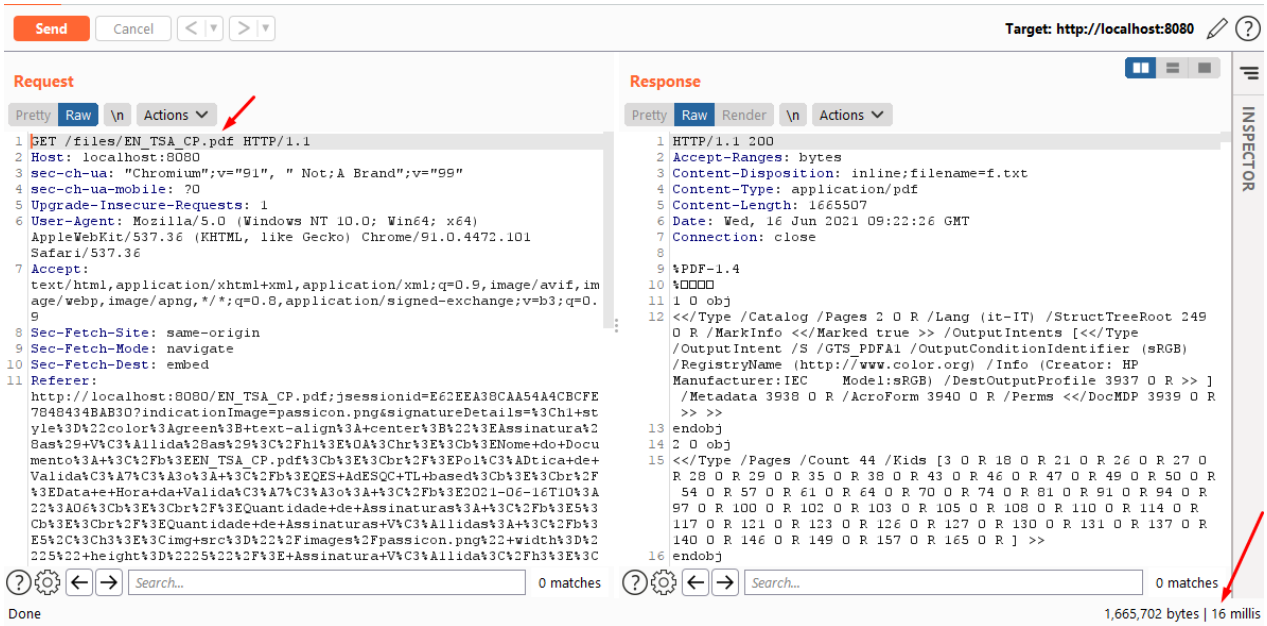


Figura 6.7: Tempo de geração do relatório básico.

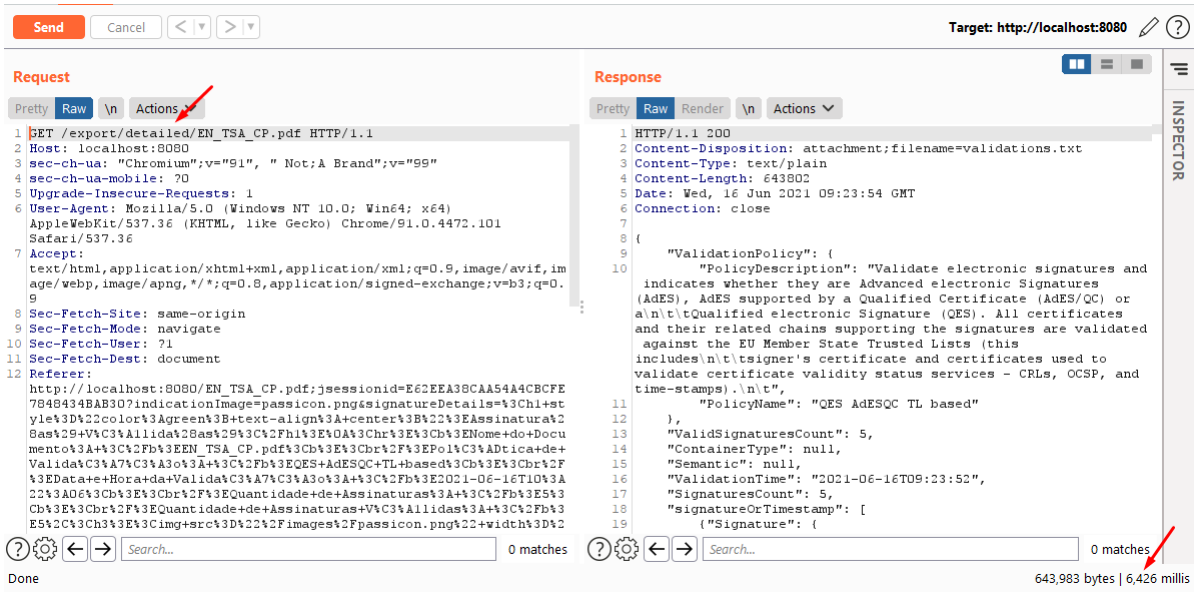


Figura 6.8: Tempo de geração do relatório avançado.

6.2 Erros Identificados

Foram identificados alguns erros de formatação de baixa relevância, são eles:

- **Assinaturas com caracteres especiais:** Caso o ficheiro possua assinaturas com nomes com caracteres especiais, o ficheiro JSON com a validação realizada pelo *DSS Bundle* retorna estes nomes com o *enconding* com *strings* ininteligíveis, Figura 6.9.

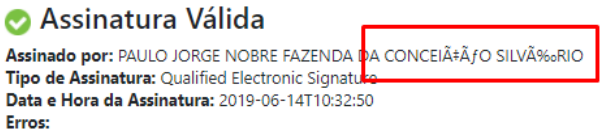


Figura 6.9: Má formatação com nomes de assinaturas com caracteres especiais.

- **Nome do ficheiro com caracteres especiais:** O nome do ficheiro a ser enviado não pode conter caracteres especiais, pois não foi implementado na REST API o tratamento do nome do ficheiro. Desta forma, o ficheiro PDF não consegue ser carregado, apenas a validação, conforme destaca a Figura 6.10.

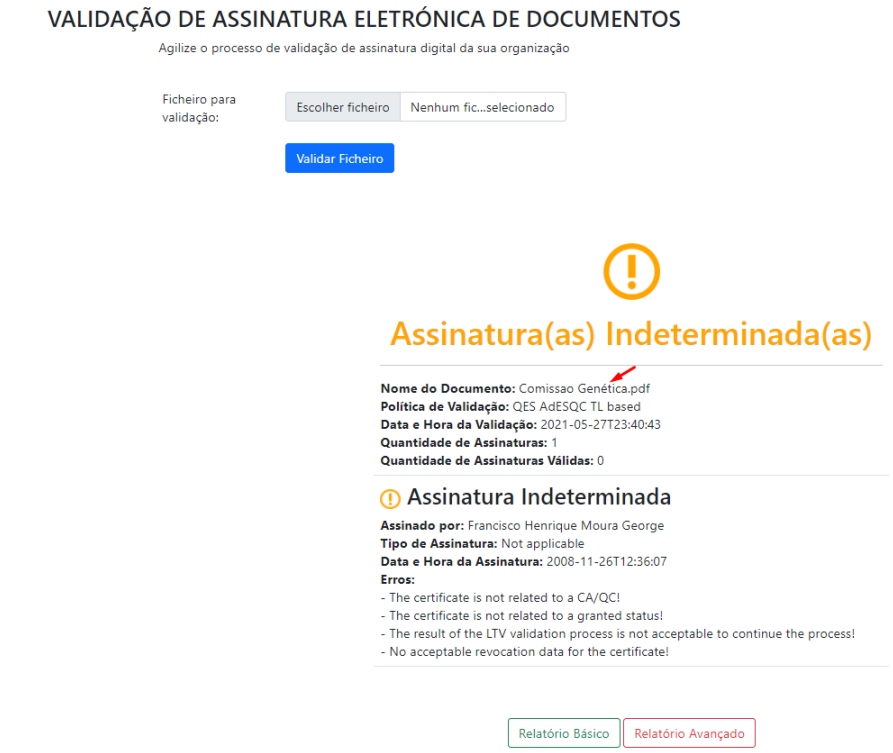


Figura 6.10: Má formatação devido ao nome do ficheiro conter caracteres especiais.

- **Nome do ficheiro muito longo:** Caso o ficheiro tenha um nome muito longo, a formatação da apresentação dos resultados no HTML pode vir a ficar mal ajustada, Figura 6.11.

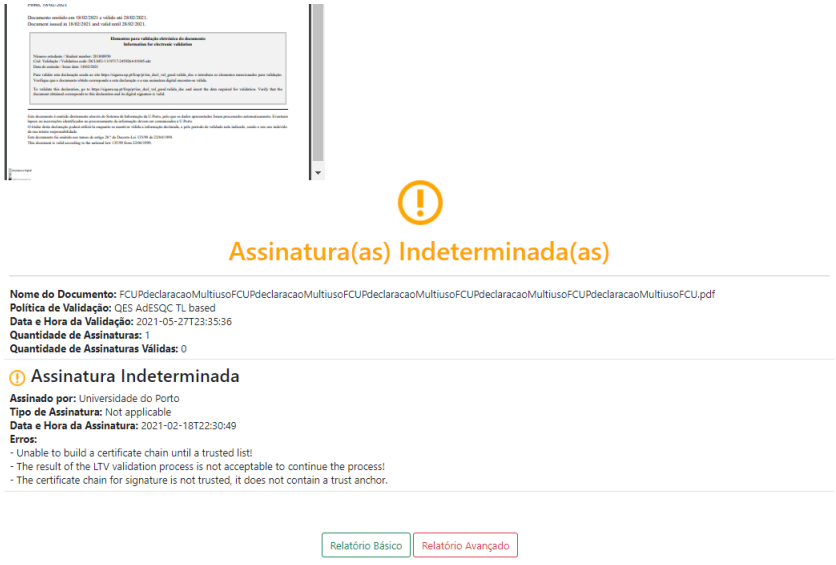


Figura 6.11: Má formatação devido ao nome muito longo do ficheiro.

6.3 Segurança Aplicacional

Por fim, do ponto de vista de segurança aplicacional, com a ferramenta Sonarqube não foram encontradas vulnerabilidades no código desenvolvido, a não ser nas bibliotecas de terceiros, o que por si só não apresentam vulnerabilidades de grande severidade ou impacto, visto que estas funções não estão a ser utilizadas na aplicação.

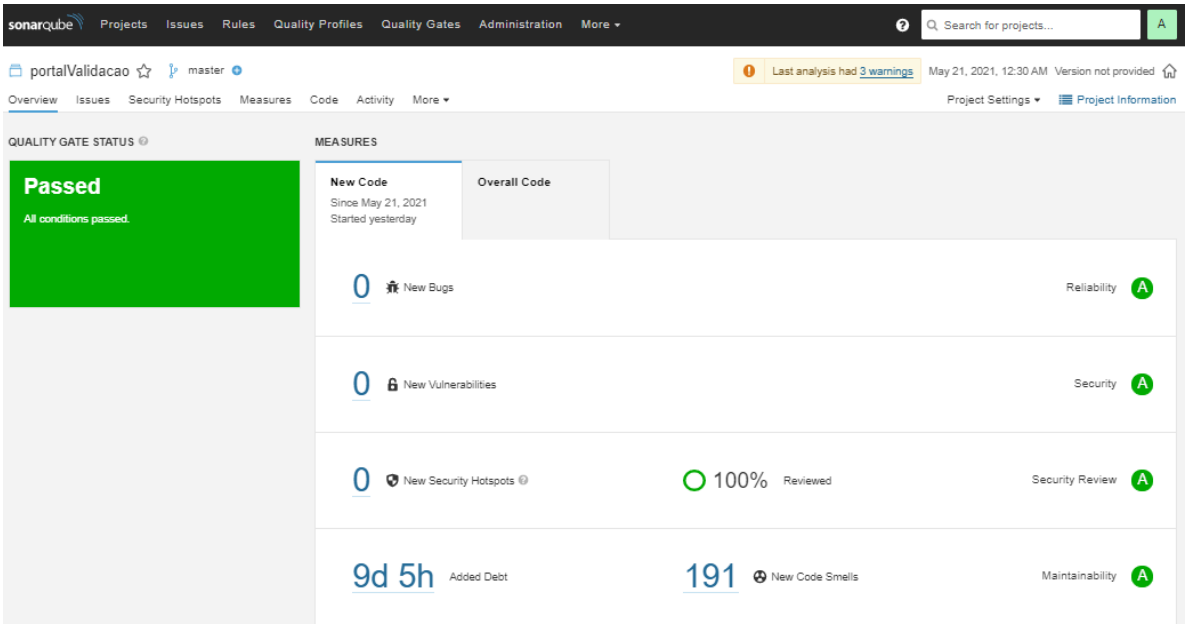


Figura 6.12: Resultado após classificação das bibliotecas como falso-positivos.

Capítulo 7

Conclusão

7.1 Resumo da Solução

Com a criação, difusão e utilização das assinaturas digitais, o seu uso tem se tornado cada vez maior, seja por parte de empresas para autenticar documentos, seja por parte dos utilizadores. Apesar da facilidade de assinar um documento utilizando o cartão do cidadão por exemplo, muitos utilizadores não possuem conhecimento tecnológico para interpretar o resultado de uma validação de uma assinatura. É através desta observação que a Loqr constatou a necessidade da criação de uma aplicação que permita aos utilizadores compreender os resultados de forma visualmente apelativa e imediata.

O protótipo desenvolvido para este trabalho permite ao utilizador através de uma interface simples e intuitiva, submeter um ficheiro e obter o resultado da validação da assinatura do documento, caso exista, de maneira clara e objetiva. As assinaturas são classificadas individualmente (válida, inválida ou indeterminada) e também de forma global. O uso de ícones e cores com forte tonalidade chamam a atenção para o resultado final. O uso da cor vermelha, psicologicamente está associada com perigo, desta forma foi escolhida para assinaturas inválidas. A cor amarela indica atenção, e foi utilizada para assinaturas indeterminadas. E por fim, a cor verde é associada ao sentimento de esperança e plenitude, e foi utilizada para caracterizar assinaturas válidas.

Ao final da validação, o utilizador tem a visualização do PDF enviado, e também consegue distinguir o resultado de cada assinatura e perceber se o documento é considerado aceitável como um todo. Para além disto, informações como: nome do ficheiro, política de validação, data e hora da validação, signatários do documento, etc., são indicados. Caso o utilizador queira consultar todo o processo de validação do *DSS Bundle*, é possível realizar o *download* do relatório básico e avançado.

7.2 Objetivos Concluídos

Dos 3 principais objetivos definidos na proposta deste trabalho, todos foram atendidos, a saber: o desenvolvimento de um portal de validação de documentos assinados com uma representação visual simplificada da validação da assinatura, a visualização do documento submetido, além da possibilidade de obter um relatório básico e detalhado de todo o processo de validação ocorrido. A aplicação resultante deste projeto atendeu as expectativas, gerando resultados visualmente e tecnicamente satisfatórios, e permite alargar a usabilidade de ficheiros do tipo PDF assinados digitalmente, visto que é necessário o utilizador possuir qualquer dispositivo com um navegador *web* para realizar a validação. Se pode afirmar que qualquer pessoa com conhecimentos limitados desta tecnologia pode utilizá-la para aferir a validade do documento e decidir a aceitação dos mesmos.

7.3 Desafios

Durante o desenvolvimento deste projeto foram encontrados alguns desafios que impactaram de forma superficial na execução, tais como:

- **Modo de envio do ficheiro para validação:** Foi necessário perceber o modo que o ficheiro PDF é enviado para o *DSS Bundle* para ser validado. Após leitura da extensa documentação, foi possível identificar que o ficheiro deveria ser codificado para Base 64 e inserido na chave `bytes` do ficheiro `getDataToSign.json` pré-formatado da solução.
- **Formatação dos erros das assinaturas:** Outro desafio encontrado foi durante o processo de obtenção dos erros da assinaturas. Caso a linha da chave do JSON possuísse uma vírgula, o `for loop` tratava a vígula como uma nova linha, deixando o texto mal formatado para o utilizador. Para resolver este problema, foi utilizado no código um tratamento que substituia a vírgula por `<span>&#44;</span>` e que é interpretado pelo navegador como vírgula.
- **Análise Estática de Código com o Sonarqube:** Ao avaliar a parte da segurança aplicacional, foi possível perceber que os ficheiros em formato .JS não estavam a ser analisados. A partir disto, foi realizada uma análise para identificar o problema e descobriu-se que era necessário instalar o NodeJS no portátil e passar através da linha de comando o `path` para o ficheiro, neste caso, `-Dsonar.nodejs.executable="PATH"`.

7.4 Trabalho Futuro

Devido ao ficheiro JSON com as validações realizadas retornar os erros das assinaturas com diferentes tipos de variáveis de modo dinâmico, não foi possível realizar em tempo útil todas

combinações para tradução das chaves do JSON de Inglês para Português. Deste modo, como sugestão de trabalho futuro, pode-se obter a lista de todos os erros na documentação do DSS Bundle, realizar as combinações e traduções, e incorporar no código desenvolvido. Segundo a documentação do *DSS Bundle* também é possível incluir Listas Confiáveis (*Trusted Lists*) e outros Certificados de Autoridade (CA) fora dos pré-definidos, desta forma, caso seja necessário, fica a critério da Loqr realizar esta verificação e implementação. Por fim, a implementação de uma área de *drag and drop* para o utilizador colocar o ficheiro para validação seria um *add-on* mais cômodo e visualmente mais amigável.

7.5 Considerações Finais

Através deste projeto foi possível obter um embasamento teórico mais amplo relativamente sobre assinaturas digitais, a sua implementação e o processo de validação. Desde a apresentação da proposta deste projeto fiquei interessado em perceber o mecanismo de desenvolvimento de uma REST API e a sua aplicação num cenário real. O uso de diversas tecnologias como Docker, Java e *Spring Boot* para o desenvolvimento do protótipo, também foram fundamentais para aprender e relembrar alguns conceitos de lógica e programação.

Conforme mencionado anteriormente, atualmente não foi identificado nenhum outro *site* que permita o utilizar validar as assinaturas contidas num ficheiro e o resultado seja apresentado de forma simples e objetiva, ou seja, de uma forma não muito técnica. Por meios dos resultados obtidos nos diferentes cenários de testes, acredito que através da implementação e divulgação deste projeto no *site* da Loqr, a quantidade de acessos possa subir consideralmente, devido a facilidade de uso e a clareza das informações apresentas ao utilizador, tornando assim, numa referência de validação de documentos assinados digitalmente.

Apêndice A

Notação Matemática Básica

$\mathbb{N}$: Conjunto dos números naturais
$\mathbb{Z}$: Conjunto dos números inteiros
$\mathbb{Q}$: Conjunto dos números racionais
$\mathbb{R}$: Conjunto dos números reais
$\perp$: Contradição, perpendicular, ortogonal, coprimo, independente ou elemento inferior
$\in$: Pertence (faz parte de)
$\notin$: Não Pertence (não faz parte de)
$\subset$: É um subconjunto de
$\cup$: União
$\cap$: Interseção
$\emptyset$: Vazio
$\rightarrow$: Implicação à direita
$<$: Menor que
$>$: Maior que
$\leq$: Menor ou igual que
$\geq$: Maior ou igual que
$=$: É igual a
e^x: exponencial de x
$\ln y$: logaritmo de y

A.1 Letras Gregas

κ: Kappa
Σ: Sigma

Apêndice B

User Stories - Gherkin

O objetivo deste apêndice é evidenciar os chamados *user stories* que foram utilizados para realizar a implementação da solução, além de ser um complemento para a criação dos requisitos do Capítulo 4. Os *user stories* têm como objetivo descrever cenários de aceitação de forma clara e transparente, tanto para o desenvolvedor como para o dono do projeto, e é composto por cinco itens:

- *Scenario*: O comportamento que é suposto acontecer;
- *Given*: O início do cenário;
- *When*: A ação específica que o utilizador vai realizar;
- *Then*: Resultado esperado após a ação do utilizador;
- *And*: Para realizar mais de uma operação.

Estes cinco itens devem ser seguidos de forma que todo o desenvolvimento seja de fato implementado de acordo com o que foi elaborado com o *Product Owner* da Loqr.

B.1 Features

Feature: Digital Signature Validation Portal
Scenario: As an User, I want to access Digital Signature Validation Portal.
Given that I am on "landing-page"
When I "use-top-bar"menu
And I "access-portal-de-validacao-de-assinaturas-digitais"
Then I expect to be on "portal-de-validacao-de-assinaturas-digitais"

Feature: PDF File Uploader - PDF Validation
Scenario: I want to validate PDF is not corrupted.
Given that I "sent-pdf-file-to-validation"
When I "pre-process-pdf-file"
Then I "validate-pdf-file-is-not-corrupted"
And I "send-file-to-processing"

Feature: PDF File Uploader - Signature Validation
Scenario: I want to validate the existence of digital signature inside the uploaded PDF file.
Given that I "receive-sucess-message-from-uploaded-file"
When I "pre-process-pdf-file"
Then I "validate-pdf-file-has-signatures"
And I "validate-pdf-file-has-more-than-one-signature"
And I "send-file-to-processing"

Feature: PDF File Uploader - Signature Validation
Scenario: I want to validate the existence of no digital signature inside the uploaded PDF file.
Given that I "receive-no-sucess-message-from-uploaded-file"
When I "pre-process-pdf-file"
Then I "validate-pdf-file-has-no-signatures"
And I "validate-error-message-is-no-signatures-were-found"
And I "load-error-screen-for-user"

Feature: PDF File Uploader - Basic
Scenario: As an User, I want to upload a PDF File to validate digital signature messages.
Given that I am on "portal-de-validacao-de-assinaturas-digitais"
When I "select-pdf-file"
And I "click-send"
Then I expect "success-upload-message"
And I "validate-message-is-total-passed-digital-signature"
And I "validate-green-symbol-is-displayed"
When I "receive-unsuccessful-message"
Then I "validate-message-is-total-failed-digital-signature"
And I "validate-red-symbol-is-displayed"
When I "receive-warning-message"
Then I "validate-message-is-indeterminate-digital-signature"
And I "validate-yellow-symbol-is-displayed"
When I "click-basic-report"
Then I "validate-file-is-generated-with-basic-informations-of-digital-signature"
And "pdf-file-from-server-is-deleted"

Feature: PDF File Uploader - Advanced
Scenario: As an User, I want to upload a PDF File to validate advanced informations related to the digital signature.
Given that I am on "portal-de-validacao-de-assinaturas-digitais"
When I "select-pdf-file"
And I "click-send"
Then I expect "success-upload-message"
And I "validate-message-is-total-passed-digital-signature"
And I "validate-green-symbol-is-displayed"
When I "click-advanced-report"
Then I "validate-file-is-generated-with-advanced-informations-of-digital-signature"
And "pdf-file-from-server-is-deleted"

Feature: Digital Signature Validation
Scenario: I want to validate digital signature from the uploaded PDF file.
Given that I "receive-sucess-message"
When I "pre-process-pdf-file"
Then I "validate-pdf-file-has-signatures"
And I "validate-pdf-file-has-more-than-one-signature"
And I "validate-timestamp-certificate"
And I "validate-root-certificate-from-trusted-list"
And I "validate-exist-more-certificates"
And I "validate-ocsp-certificate"
And I "validate-crl-certificate"

Bibliografia

[1] Reshma Afshar. Digital Certificates (Public Key Infrastructure). *Indiana State University*, Outubro 2015 [visited Janeiro 2021].

[2] ANACOM. ETSI - European Telecommunications Standards Institute. Online [visited Março 2021].

[3] European Comission. Digital Signature Service. Online [visited Março 2021].

[4] Cryptomathic. What is a Qualified Electronic Signature? Online [visited Janeiro 2021].

[5] AEDP Agencia Española Protección datos. Introduction to the Hash Function as a Personal Data Pseudonymisation Technique. *EDPS - European Data Protection Supervisor*, Outubro 2019 [visited Janeiro 2021].

[6] Christof Paar e Jan Pelzl. *Understanding Cryptography*. Springer, Berlin, Heidelberg, 2010. ISBN: 978-3-642-04101-3.

[7] Mehran Alidoost Nia et. al. An Introduction to Digital Signature Schemes. *University of Tehran* [visited Dezembro 2020].

[8] R. Gennaro et. al. Batching Schnorr Identification Scheme with Applications to Privacy-Preserving Authorization and Low-Bandwidth Communication Devices. *Springer-Verlag Berlin Heidelberg*, page 276–292, 2004 [visited Janeiro 2021].

[9] ETSI. Electronic Signatures and Infrastructures (ESI); Signature Validation Procedures and Policies. (V1.1.2 2012-10), [visited Fevereiro 2021].

[10] ETSI. Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 1: PAdES Overview - a framework document for PAdES. (V1.1.1 2009-07), [visited Janeiro 2021].

[11] ETSI. Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 4: PAdES Long Term - PAdES-LTV Profile. (V1.1.1 2009-07), [visited Janeiro 2021].

[12] Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012. ISBN: 9781139012843.

[13] NAMIRIAL GmbH. A guide on eIDAS 910/2014. Namirial DTM Solution for Legally Compliant e-Signatures. Online [visited Janeiro 2021].

[14] GNS - Gabinente Nacional de Segurança. Assinatura eletrónica. Online [visited Janeiro 2021].

[15] Cátia Cristina D. R. Gomes. O impacto dos diferentes tipos de assinatura digital nas empresas do séc. xxi: Activobank e iscte-iul. Master's thesis, ISCTE-IUL, 2015 [visited Janeiro 2021].

[16] Hans-Petter Halvorsen. Web Programming. Online [visited Janeiro 2021].

[17] Blaine Hein. PKI Trust Models: Whom do you trust? Online, 2013 [visited Janeiro 2020].

[18] Network Associates Inc. *An Introduction to Cryptography*. 1990–2000 [visited Dezembro 2020].

[19] David Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner; Rev Sub edition, 1996. ISBN: 978-0684831305.

[20] Deepak Kumar. *Best Practices for Building Restful Web Services*. Infosys [visited Janeiro 2021].

[21] Luxtrust. What is a Registration Authority (RA)? Online, 2020 [visited Dezembro 2020].

[22] Debdayal Mandol. Basics of Digital Certificates and Certificate Authority. Online [visited Janeiro 2021].

[23] Digital Single Market. eIDAS made easy. A Quickstart Guide to Electronic Identification (eID) and Trust Service Solution for Business. Online [visited Janeiro 2021].

[24] Thomson Reuters. Electronic Signatures Plataforms Key Contractual Issues. 2017 [visited Janeiro 2021].

[25] Joel Tiago Ribeiro. Cifra de Vigenère. Técnicas de Data Mining para Criptoanálise. Master's thesis, Universidade do Minho [visited Fevereiro 2021].

[26] Scand. Pros and Cons of Using Spring Boot. Online, Junho 2020 [visited Março 2021].

[27] Bruce Schneier. *Applied Cryptography - Protocols, Algorithms, and Source Code in C*. Wiley, 1996. ISBN: 978-1-119-09672-6.

[28] Signicat. Pades. Online, Agosto 2017 [visited Janeiro 2021].

[29] CA Technologies. Creating REST APIs to Enable Your Connected World. Online, Janeiro 2017 [visited Janeiro 2021].

[30] Vladimir Omar Calderon Yaksic. A Study on Hash Functions for Cryptography. *GIAC*, 2003 [visited Janeiro 2021].

[31] Kostas Zoto and Andreas Litke. Cryptography and Encryption. *Dept. of Applied Informatics - University of Macedonia* [visited Dezembro 2020].