Universidade de Brasília – UnB
Faculdade de Ciências e Tecnologias em Engenharia - FCTE
Engenharia de Software

UnBSign: Desenvolvimento de uma API de Assinatura Digital Integrada a Web e PKI

Autor: Sidney Fernando Ferreira Lemes
Orientador: Prof. Dr. John Lenon Cardoso Gardenghi

Brasília, DF
2025

Sidney Fernando Ferreira Lemes

UnBSign: Desenvolvimento de uma API de Assinatura Digital Integrada a Web e PKI

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade de Ciências e Tecnologias em Engenharia - FCTE

Orientador: Prof. Dr. John Lenon Cardoso Gardenghi

Brasília, DF

2025

Sidney Fernando Ferreira Lemes
UnBSign: Desenvolvimento de uma API de Assinatura Digital Integrada a Web e PKI/ Sidney Fernando Ferreira Lemes. – Brasília, DF, 2025-
99 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. John Lenon Cardoso Gardenghi

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade de Ciências e Tecnologias em Engenharia - FCTE , 2025.

1. Assinatura Digital. 2. Criptografia. I. Prof. Dr. John Lenon Cardoso Gardenghi. II. Universidade de Brasília. III. Faculdade de Ciências e Tecnologias em Engenharia. IV. UnBSign: Desenvolvimento de uma API de Assinatura Digital Integrada a Web e PKI

CDU 02:141:005.6

Sidney Fernando Ferreira Lemes

UnBSign: Desenvolvimento de uma API de Assinatura Digital Integrada a Web e PKI

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 18 de fevereiro de 2025 – Data da aprovação do trabalho:

Prof. Dr. John Lenon Cardoso Gardenghi
Orientador

Prof. Dr. Bruno César Ribas
Convidado 1

Prof. Dr. Tiago Alves da Fonseca
Convidado 2

Sérgio de Almeida Cipriano Júnior
Convidado 3

Brasília, DF
2025

Agradecimentos

À medida que chego ao final desta jornada chamada graduação, gostaria de expressar minha gratidão a todos que contribuíram para este momento. Primeiramente agradeço a Deus. Também agradeço aos amigos que fiz ao longo do caminho e que pretendo levar por muitos anos. Agradeço também os professores e ao meu orientador, John Lennon, cuja orientação e apoio foram fundamentais nesta etapa final. Em especial, quero agradecer aos meus pais, Sidney e Adaide, que tanto se esforçaram e caminharam debaixo do sol para que eu pudesse caminhar na sombra. Sem o apoio e sacrifício deles, essa conquista não seria possível.

Resumo

Este trabalho tem como objetivo o desenvolvimento de uma API (*Application Programming Interface*, ou Interface de Programação de Aplicações) para assinatura digital, com a finalidade de ser integrada futuramente à Universidade de Brasília (UnB). A proposta busca contribuir para a segurança dos processos administrativos da instituição, além de promover a adoção de uma solução padronizada de assinatura digital, visando à maior uniformidade e adesão por parte da comunidade acadêmica. A API foi concebida de modo a alinhar-se às normas legais vigentes, garantindo conformidade com as exigências regulatórias aplicáveis.

Como parte do projeto, foi desenvolvida uma aplicação web simples, juntamente com uma PKI (*Public Key Infrastructure*, ou infraestrutura de chaves públicas), com o objetivo de demonstrar o funcionamento da API. Essa implementação possibilita a visualização prática de sua integração e usabilidade, além de fornecer um ambiente controlado para testes e validação das funcionalidades de assinatura digital.

Para o desenvolvimento da solução, foram realizadas análises das legislações pertinentes, avaliações das tecnologias disponíveis para assinatura e certificação digital, e aplicadas técnicas de Engenharia de Software. Dessa forma, a expectativa é que a solução possa servir como uma alternativa viável para a instituição, contribuindo para a modernização e padronização de seus fluxos de trabalho.

Palavras-chave: Assinatura Digital. Criptografia. Certificado digital.

Abstract

The aim of this work is to develop an API (*Application Programming Interface*) for digital signatures, to be integrated into the University of Brasília (UnB) in the future. The proposal seeks to contribute to the security of the institution's administrative processes, as well as promoting the adoption of a standardized digital signature solution, with a view to greater uniformity and adherence by the academic community. The API was designed to align with current legal standards, ensuring compliance with applicable regulatory requirements.

As part of the project, a simple web application was developed, along with a PKI (Public Key Infrastructure), with the aim of demonstrating how the API works. This implementation enables practical visualization of its integration and usability, as well as providing a controlled environment for testing and validating the digital signature functionalities.

To develop the solution, the relevant legislation was analyzed, the available digital signature and certification technologies were evaluated and software engineering techniques were applied. In this way, the solution is expected to serve as a viable alternative for the institution, contributing to the modernization and standardization of its workflows.

Key-words: Digital Signature. Cryptography. Digital Certificate.

Lista de ilustrações

Lista de tabelas

Sumário

[1] Os tópicos *Object Identifier (OID)*, Novas Versões, Proteção contra Alterações Indevidas, Entidade Criadora da Política de Assinatura, Período para Assinatura, Raiz Confiável, Conjunto de Políticas de Certificado Aceitável e Forma de Verificação do Status da Cadeia de Certificação (Revogação) não se aplicam ao contexto real, sendo utilizados somente de forma didática para este trabalho.

1 Introdução

Observa-se que a interação entre os diversos setores da sociedade tem incorporado, em diferentes graus, elementos digitais. “Nos últimos anos, a tecnologia vem avançando rapidamente, impulsionada por tecnologias emergentes e mudanças culturais”(Serasa, 2024). Na parte que diz respeito à digitalização de processos e modernização de práticas administrativas, a tecnologia de assinatura eletrônica, também chamada de assinatura digital, surge como uma ferramenta para autenticação de documentos que tem potencial para ser mais segura e eficiente. “As assinaturas digitais são uma ferramenta poderosa e agora são aceitas como legalmente vinculantes em muitos países; elas podem ser usadas para certificar contratos ou autenticar documentos, para autenticação de pessoas físicas ou jurídicas e como componentes de protocolos mais complexos.”(KATZ, 2010) . Embora ela possa, em seu mais alto nível de confiabilidade, utilizar criptografia, é importante ressaltar que, apesar de ser um método seguro, não está isento de vulnerabilidades. A segurança desses algoritmos depende de fatores como a complexidade das chaves, a robustez dos algoritmos utilizados, e a proteção contra ataques cibernéticos que possam comprometer as chaves privadas, tornando necessário um acompanhamento contínuo e atualizações para manter o nível de proteção desejado.

> No mundo digital, com frequência deseja-se indicar o dono ou o criador de um documento ou deixar claro que alguém concorda com o conteúdo de um documento. A assinatura digital é uma técnica criptográfica que cumpre essas finalidades no mundo digital (KUROSE; ROSS, 2021).

Ainda que a Universidade de Brasília (UnB) já adote outras formas de gestão de documentos, como o Sistema Eletrônico de Informações (SEI), como é apresentado em (UnB, 2024), a implementação dessa tecnologia poderia complementar as práticas existentes, proporcionando uma solução interna alinhada às necessidades específicas da instituição.

Considerando o uso do Sistema Integrado de Gestão (SIG-UnB), “um conjunto de ferramentas online adotado para unificar e modernizar os diversos sistemas internos de gestão da instituiçã” (SIGUnB, 2020), propõe-se o desenvolvimento de uma interface de programação de aplicações (API) para assinatura digital, que futuramente possa ser integrada ao Sistema Integrado de Gestão de Atividades Acadêmicas (SIGAA). Tal integração pode fortalecer a eficiência nos processos administrativos, ampliando a conectividade e a uniformidade entre os sistemas, e contribuindo para uma gestão mais integrada na universidade, dado que o SIGAA, como citado acima, tem como objetivo unificar os sistemas internos e proporcionar maior coesão nas operações acadêmicas e administrativas.

1.1 Justificativa

A implementação de uma ferramenta para assinatura digital na UnB é relevante, uma vez que "[...] as assinaturas eletrônicas são válidas e reconhecidas legalmente" (GOV.BR, 2020). A Lei nº 14.063, de 23 de Setembro de 2020, dispõe sobre as regras para uso das assinaturas eletrônicas nas interações entre pessoas e instituições privadas com os entes públicos e entre os próprios órgãos e entidades públicas (BRASIL, 2020).

O desenvolvimento de uma API de assinatura digital no contexto da Universidade de Brasília (UnB) apresenta-se como uma alternativa estratégica às soluções atualmente disponíveis.O SEI, mencionado anteriormente, não apresenta diretrizes claras sobre o uso por discentes em UnB (2016), onde fala sobre assinatura eletrônica, além de não ser amplamente adotado por esse público. Como consequência, os estudantes podem recorrer a ferramentas externas, o que pode eventualmente representar riscos à segurança e à integridade dos documentos assinados. "O processo de assinatura digital pode ser feito diretamente em programas de edição de texto e também em PDFs, mas há um risco nessa opção. Isso porque, a gestão desses documentos é precária e podem se perder os arquivos."(Equipe TOTVS, 2024). Já o assinador do Gov.br, apesar de ser uma solução gratuita e oferecer um nível avançado de segurança, mesmo sendo do governo federal, impõe uma dependência de uma plataforma externa à instituição.

Diante desse cenário, o desenvolvimento de uma solução própria, integrada ao ecossistema tecnológico da UnB, pode se configurar como uma alternativa para auxiliar na integração digital da instituição. Um sistema de assinatura digital interno teria o potencial de impactar positivamente a experiência da comunidade acadêmica e administrativa.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver um *MVP* (*Minimum Viable Product*), ou Produto Mínimo Viável, de uma API REST (*Application Programming Interface - Representational State Transfer*) para assinatura digital, em conformidade com a legislação vigente. Um *MVP* é a versão mais simples de um produto, criada para testar sua viabilidade com o mínimo de esforço, como explica Lean Startup Glossary (2015). Além da API, será desenvolvida uma aplicação web (*web app*) e uma infraestrutura simples de chaves públicas integradas a ela, demonstrando seu funcionamento na prática.

Para alcançar esse objetivo, serão adotadas abordagens de Engenharia de Software, incluindo metodologias ágeis e práticas de integração contínua (*Continuous Integration* – CI). O desenvolvimento será guiado pela criação de épicos com histórias de usuário e pela elicitação de requisitos, garantindo alinhamento com as necessidades dos usuários e

as melhores práticas de mercado.

1.2.2 Objetivos Específicos

- Investigar a legislação pertinente ao uso de assinatura digital;
- Analisar as tecnologias disponíveis para implementação de assinatura digital, certificação digital e manipulação de documentos;
- Empregar técnicas de Engenharia de Software para projetar e desenvolver um software de assinatura digital;

1.2.3 Organização do Documento

A organização deste documento segue uma estrutura metodológica que objetiva facilitar a compreensão e o desenvolvimento. O conteúdo está distribuído nos seguintes capítulos, além da introdução:

- **2 - Referencial Teórico**: apresenta uma revisão da literatura sobre temas fundamentais para o desenvolvimento do sistema, como segurança da informação, criptografia, certificados digitais, infraestrutura de chaves públicas e as legislações relacionadas. A partir deste referencial, são discutidos os conceitos que embasam o projeto, incluindo a infraestrutura necessária para a implementação de assinaturas digitais válidas e seguras;
- **3 - Metodologia**: são detalhadas as abordagens adotadas para a pesquisa e o desenvolvimento do sistema. A metodologia inclui a classificação dos tipos de pesquisa, a abordagem adotada para a pesquisa bibliográfica e a metodologia de desenvolvimento utilizada, com ênfase na gerência de configuração e evolução de software. Também são apresentados os fluxos de desenvolvimento e o cronograma do projeto, proporcionando uma visão clara das etapas a serem seguidas;
- **4 - Desenvolvimento do Software**: são abordadas as histórias de usuário e os requisitos do sistema, além das tecnologias utilizadas na construção da API UnB-Sign, *Web App* e API PKI (*Public Key Infrastructure*, ou Infraestrutura de Chaves Públicas). A arquitetura das soluções é detalhada, incluindo a arquitetura da API REST, da *Web App* e da API PKI, com uma explicação sobre a estrutura e funcionamento de cada componente. O objetivo é ilustrar como as tecnologias e arquiteturas escolhidas se integram para atender aos requisitos do sistema de assinatura digital;
- **5 - Resultados**: apresenta os resultados obtidos durante o desenvolvimento do sistema. Nele, são exibidos diagramas de arquitetura funcional que ilustram a integração dos componentes do sistema. Além disso, são discutidos em detalhe a API

UnBSign, a API PKI e o *Web App*, com foco nas funcionalidades implementadas, como a assinatura e validação de documentos PDF, a emissão e controle de certificados digitais, e os endpoints da API.

- **6 - Conclusão**: oferece uma análise dos resultados alcançados e das contribuições do projeto. São discutidas as considerações finais sobre a implementação e as dívidas técnicas, assim como sugestões de melhorias para futuras versões do sistema. Este capítulo visa fornecer uma visão crítica do trabalho realizado e indicar os próximos passos para o aprimoramento da solução;

2 Referencial Teórico

O referencial teórico está estruturado em seções que abordam aspectos essenciais da segurança da informação. Inicia-se com uma introdução ao conceito de segurança, seguida por uma análise breve sobre criptografia, abrangendo suas principais variantes, como a criptografia assimétrica e as funções de *hash*. Em seguida, são explorados os certificados digitais, com uma explicação detalhada sobre seus diferentes tipos, além das Infraestruturas de Chaves Públicas. Posteriormente, discute-se a legislação relacionada às chaves públicas e à assinatura eletrônica, com destaque para a ICP-Brasil(Infraestrutura de Chaves Públicas Brasileira) e as características das assinaturas eletrônicas. A seção se encerra com uma análise aprofundada da assinatura digital, consolidando a compreensão do tema, além de abordar conceitos de API e *Web App*.

2.1 Segurança da Informação

O uso da assinatura digital está intimamente relacionado à segurança da informação, a qual pode ser dividida no atendimento aos seguintes requisitos: confidencialidade, autenticação, não repúdio e integridade (TANENBAUM; FEAMSTER, 2021). Considerando que mensagem no contexto deste trabalho é definida como o conjunto de dados que está sendo transmitido, estes requisitos, também explicados por Tanenbaum e Feamster (2021), podem ser definidas como:

- *Sigilo ou Confidencialidade*: A mensagem deve ser cifrada com o intuito de que somente o remetente e o destinatário entendam seu conteúdo;
- *Autenticação*: A identidade do remetente e do destinatário precisam ser confirmadas um pelo outro;
- *Não repúdio*: Deve-se certificar que a mensagem recebida é de fato legítma, ou seja, quando uma mensagem é enviada ao destinatário, este pode provar que o suposto remetente de fato enviou a mensagem;
- *Integridade*: refere-se à garantia de que a mensagem enviada não foi alterada de forma não autorizada durante sua transmissão. Isso significa que é fundamental assegurar que o conteúdo enviado seja exatamente o mesmo que o recebido, sem ter sido modificado por acidente, interferência de terceiros ou qualquer outro problema no caminho da comunicação.

A proteção de uma mensagem pode ser realizada por meio da criptografia, que emprega chaves para garantir a segurança da informação. Dependendo do contexto, funções de *hash* criptográficas também podem ser utilizadas para assegurar a integridade da mensagem, ou seja, garantir que ela não tenha sido alterada durante o processo de transmissão. A criptografia pode ser aplicada tanto em sistemas de chaves simétricas quanto assimétricas, sendo que, no caso de chaves públicas, a autenticidade da chave pode ser validada por meio de certificados digitais.

2.1.1 Criptografia

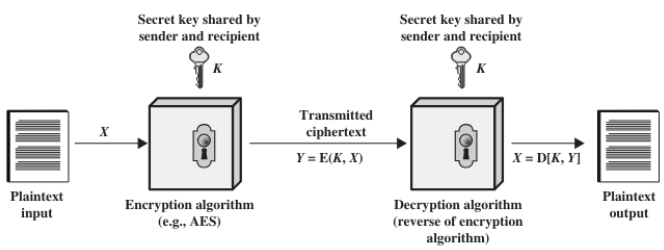
A criptografia pode ser entendida na própria origem da palavra: "do Grego *kryptós* 'escondido', mais *graphé*, 'escrita'. A rigor, qualquer mensagem que exija um processo de decifração para ser entendida e que seja escrita cai nessa definição" (Origem da Palavra, 2014). Sendo assim, criptografia é a área que utiliza técnicas para proteger informações por meio de algoritmos de cifração.

Como explica Tanenbaum e Feamster (2021), a ideia é que a mensagem a ser cifrada, conhecida como **texto em claro**, é transformada por meio de uma função parametrizada por uma **chave** (algoritmo de codificação), processo denominado **cifração**, ou processo de criptografia. Posteriormente, a saída desse processo, chamada de **texto cifrado**, é transmitida ao destinatário. O destinatário utiliza uma chave (algoritmo de decodificação - que é essencialmente o reverso do algoritmo de codificação) para converter os dados criptografados de volta para sua forma original, permitindo sua compreensão, em um processo chamado **decifração**.

Quando remetente e destinatário utilizam a mesma chave tanto para cifrar quanto para decifrar a mensagem, trata-se de criptografia simétrica. Em contraste, na criptografia assimétrica, são empregadas duas chaves distintas: uma para cifrar e outra diferente para decifrar a mensagem.

Na Figura 1 está um modelo simplificado do processo de criptografia simétrica:

Figura 1 – Processo de criptografia e decriptografia simétrica



O texto em claro X é criptografado usando a chave K, resultando no texto cifrado. Em seguida, o texto cifrado é decifrado usando a mesma chave K, gerando o texto de saída compreensível pelo remetente.

Fonte: Stallings (2013).

Existem diversas técnicas e algoritmos de criptografia simétrica, mas que fogem ao

escopo deste trabalho, visto que ao tratar de assinatura digital, estamos mais interessados em criptografia assimétrica.

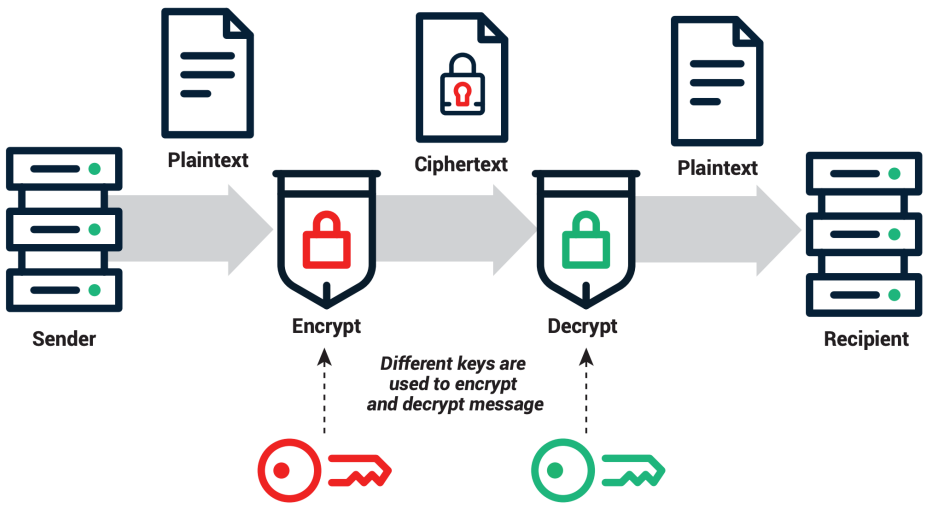
2.1.2 Criptografia assimétrica

A criptografia assimétrica, também chamada de criptografia de chave pública, foi a maior e talvez a única verdadeira revolução na área da criptografia, afirma Stallings (2013). Este modelo de criptografia utiliza duas chaves distintas nos processos de criptografia e decriptografia, por isso ela é assimétrica, ao contrário da criptografia simétrica que utiliza somente uma chave. Como será discutido mais a frente, o uso de criptografia assimétrica tem resultados em como os requisitos de confidencialidade, distribuição de chaves e autenticação são aplicados.

As chaves assimétricas são definidas por NIST (2013) como duas chaves associadas, uma pública e outra privada, que são usadas para executar ações que se complementam, como cifrar e decifrar informações ou gerar e verificar assinaturas. Além disso, como explica Kurose e Ross (2021), com apenas o algoritmo criptográfico e a chave de criptografia (pública ou privada), a chave de decriptografia não pode ser derivada de forma viável computacionalmente.

Além das chaves assimétricas, os outros componentes de um processo de criptografia assimétrica são a mensagem, o algoritmo de criptografia, o texto cifrado e o algoritmo de decriptografia. Na Figura 2 é possível ver o processo de criptografia assimétrica e seus componentes.

Figura 2 – Componentes e processo de transmissão de uma mensagem utilizando criptografia assimétrica



Fonte: SECTIGO (2020)

O processo acima opera basicamente da seguinte forma:

1. Os participantes geram um par de chaves: uma pública e uma privada;
2. Cada participante guarda sua chave privada em segredo, enquanto a chave pública pode ser amplamente divulgada;
3. Quando um participante deseja enviar uma mensagem para o outro, a chave pública do participante destinatário é usada para cifrar a mensagem, garantindo que apenas ele possa descriptografá-la utilizando sua chave privada;
4. O participante destinatário utiliza sua chave privada para decifrar a mensagem recebida. Como a chave privada é mantida em segredo, somente o destinatário pode realizar essa operação com sucesso.

O processo explicado acima também pode ser feito da seguinte forma: o remetente utiliza sua chave privada para cifrar a mensagem, e o destinatário usa a chave pública do remetente para decriptografá-la. Dessa forma, o destinatário pode verificar a autenticidade da mensagem, pois só a chave pública do remetente pode decriptografá-la com sucesso. Isso garante a autenticidade da mensagem.

2.1.3 Funções de *hash* criptográficas

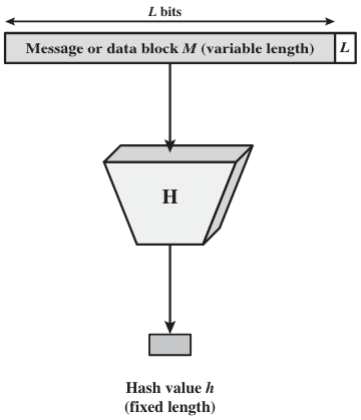
A ideia principal de uma função de *hash*, também chamada de função de resumo, é ser uma função matemática que transforma uma mensagem de entrada em uma saída única de tamanho fixo, ou seja, uma mudança na mensagem original, origina um *hash* diferente. Stallings (2013) aponta que uma função de *hash* é considerada boa se, dado um grande conjunto de entradas, as saídas produzidas são distribuídas uniformemente e aparentemente aleatórias.

Para aplicação em segurança, as funções de *hash* utilizadas são chamadas de funções de *hash* criptográficas. Segundo Kurose e Ross (2021), uma função de *hash* criptográfica deve garantir que seja extremamente difícil em termos de processamento encontrar duas mensagens diferentes que resultem no mesmo valor de hash, fenômeno conhecido como colisão. Na Figura 3, é mostrado o diagrama que ilustra o processo de *hash* de uma mensagem com tamanho L, resultando em um *hash* h.

2.2 Certificado Digital

Outro aspecto importante no contexto da assinatura digital é ter uma forma segura para divulgar a chave pública de um titular, permitindo que outras entidades, como pessoas e empresas, possam verificar a autenticidade das chaves públicas desse titular. Ou seja, deve haver um meio de certificar se a chave pública realmente pertence ao titular alegado, garantindo que ela não foi alterada ou forjada por um terceiro mal-intencionado, e

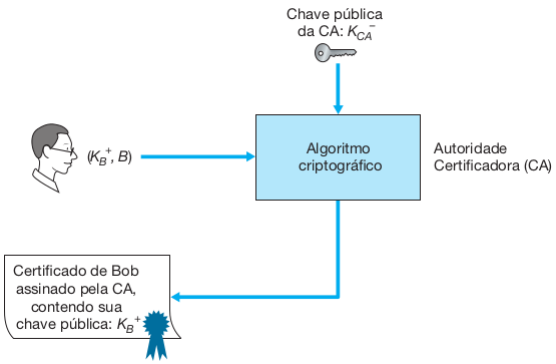
Figura 3 – Digrama básico da função de *hash* criptográfica



Fonte:Stallings (2013)

para isso existe um certificado. “A principal função de um certificado é vincular uma chave pública ao nome de um protagonista. Os certificados em si não são secretos ou protegidos” (TANENBAUM; FEAMSTER, 2021). Existem diversos tipos de cerficados, sendo que os certificados digitais são aqueles emitidos pelas chamadas Autoridades Certificadoras (CA). Uma CA verifica a identidade de uma entidade e emite um certificado digital, associando a chave pública da entidade à sua identidade verificada. A confiança na identidade está ligada à confiança na CA e em suas técnicas de verificação. A simplificação do processo de emissão de um certificado por uma CA pode ser visualizada na Figura 4.

Figura 4 – Processo de emissão de um certificado digital por uma CA.



Fonte:Kurose e Ross (2021)

Sendo assim, Souza e Neto (2017) resume que um certificado digital é um arquivo eletrônico que contém dados do indivíduo ou entidade, junto com sua Chave Pública, emitido e chancelado digitalmente por uma CA. Ele serve como prova de identificação, equivalente a documentos físicos como carteira de identidade e passaporte.

Além do aspecto de garantia da autencidade da chave pública, outro aspecto importante é o reconhecimento legal de chaves. No Brasil isso é feito por meio da Infraestrutura de Chaves Públicas Brasileiras, ICP-Brasil, que será discutida mais adiante.

2.2.1 Tipos de Certificado Digital

Conforme descrito pela ICP-Brasil, existem três categorias de certificados digitais. "A diferença de cada tipo é o nível de segurança e o modo de armazenamento das chaves" (ITI, 2020). O certificado Tipo A é designado para uso em assinaturas digitais. O Tipo S é voltado para o aumento da confidencialidade das transações. Por fim, o Tipo T, também conhecido como carimbo de tempo ou *timestamp*, é empregado para assegurar a data e a hora das transações, sendo essencial para garantir a temporalidade e a tempestividade dos documentos. Ele é frequentemente utilizado em conjunto com outros tipos de certificado.

Neste trabalho, o foco principal recai sobre o certificado Tipo A. Este é extensivamente empregado para realizar assinaturas digitais em uma ampla gama de documentos e transações eletrônicas, dentre outras possíveis aplicações. Suas funções primordiais incluem a identificação do assinante, a garantia da autenticidade da operação e a confirmação da integridade do documento assinado. Os modelos mais comuns do certificado Tipo A são o A1, com validade de um ano, e o A3, que pode ser válido por até cinco anos.

2.3 Infraestrutura de Chaves Públicas

A Infraestrutura de Chave Pública (PKI,*Public Key Infrastructure*) é um sistema que combina, políticas e procedimentos para garantir a segurança de comunicações digitais, possibilitando a autenticação, integridade e confiabilidade de dados. A PKI é composta por ferramentas que criam e gerenciam certificados digitais para diferentes entidades (usuários, organizações ou dispositivos). "A função da PKI é fornecer um modo de estruturar esses componentes e definir padrões para os vários documentos e protocolos"(TANENBAUM; FEAMSTER, 2021).

Uma PKI simples funciona como um sistema de confiança para autenticar e proteger comunicações digitais, utilizando certificados digitais. O funcionamento básico apresentado na Figura 5 é o seguinte:

1. **Criação de Pares de Chaves**: o usuário ou dispositivo gera um par de chaves (pública e privada). A chave privada é mantida em segredo, enquanto a chave pública será compartilhada.

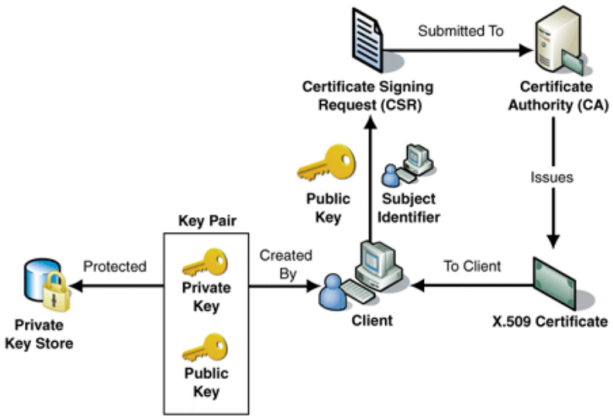
2. **Solicitação de Certificado**: O usuário envia uma Solicitação de Assinatura de Certificado (CSR, *Certificate Signing Request*) para uma Autoridade Certificadora (CA). O CSR contém informações necessárias para solicitar um certificado digital, que inclui a chave pública do solicitante e dados, como nome, organização e domínio.

3. **Emissão de Certificado**: A CA após verificar a identidade do solicitante de acordo com o nível de validação requerido, emite um certificado digital. O certificado X.509

contém informações como a identidade do proprietário, a chave pública, o período de validade e a assinatura digital da CA, que autentica o certificado e assegura que ele não foi alterado.

4. **Verificação de Validade**: Os terceiros que recebem o certificado podem verificar sua autenticidade por meio da assinatura digital da CA.

Figura 5 – Componentes e processo de uma PKI simples



Fonte: Bizagi. Disponível em: X.509 overview.

A hierarquia de PKI é uma estrutura organizada de entidades confiáveis, geralmente composta por uma Autoridade Certificadora Raiz (*Root CA*) no topo, que emite certificados para Autoridades Certificadoras Intermediárias. Essas intermediárias, por sua vez, emitem certificados para os usuários finais ou servidores. A raiz é a autoridade mais confiável, e os certificados são válidos em cadeia até a raiz, garantindo a confiança em todas as entidades da hierarquia. "Uma cadeia de certificados como essa, que volta à raiz, às vezes é chamada corrente de confiança ou caminho de certificação. A técnica é amplamente utilizada na prática"(TANENBAUM; FEAMSTER, 2021).

2.4 Legislação sobre Chaves Públicas e Assinatura Eletrônica

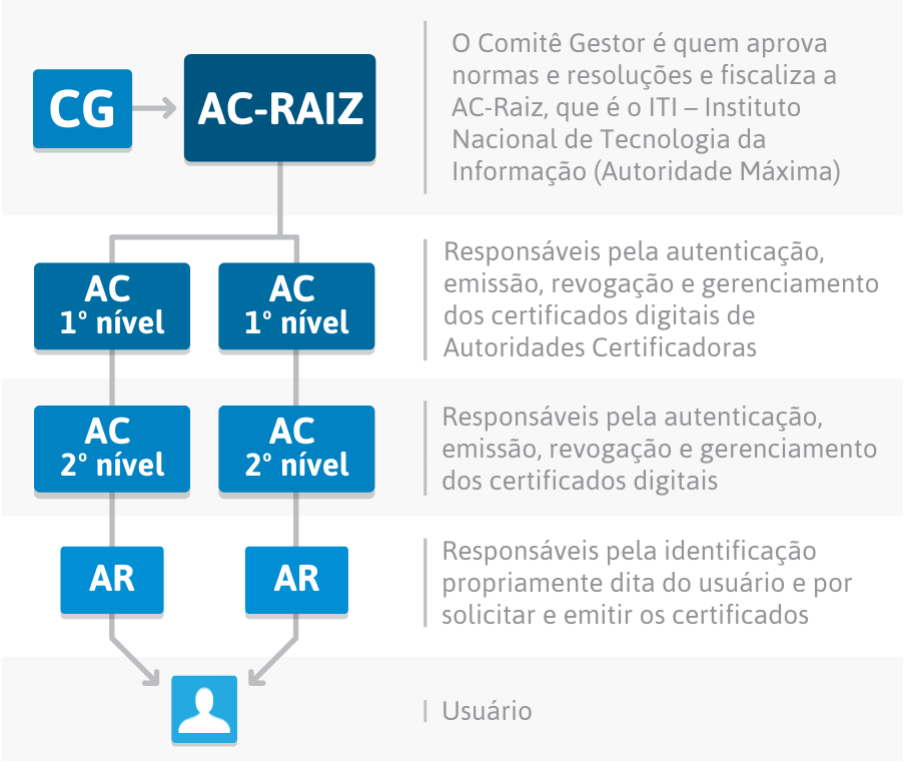
2.4.1 ICP-Brasil

A Medida Provisória 2.200-2 de 24 de agosto de 2001 institui a ICP-Brasil: "[...] para garantir a autenticidade, a integridade e a validade jurídica de documentos em forma eletrônica, das aplicações de suporte e das aplicações habilitadas que utilizem certificados digitais, bem como a realização de transações eletrônicas seguras" (BRASIL, 2001).

"A Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil) é uma cadeia hierárquica de confiança que viabiliza a emissão de certificados digitais para identificação virtual do cidadão" (ITI, 2017). O modelo brasileiro adotado para certificação é baseado em uma raiz única. A ICP-Brasil não apenas atua como Autoridade Certificadora Raiz

(CA-Raiz), mas também é responsável por credenciar e descredenciar outros participantes, supervisionar e auditar processos, emitir, distribuir, revogar e gerenciar os certificados das Autoridades Certificadoras (CAs). A ICP-Brasil é hierarquizada da seguinte forma: a CA Raiz (Autoridade Certificadora Raiz), as CAs (Autoridades Certificadoras) de primeiro nível e segundo nível, as ARs (Autoridades de Registros), e, finalmente, o usuário final. O diagrama dessa hierarquia pode ser vista na Figura 6.

Figura 6 – Hierarquia da ICP-Brasil



Fonte: Benefícios e Aplicações da Certificação Digital, 2012

2.4.2 Assinaturas Eletrônicas

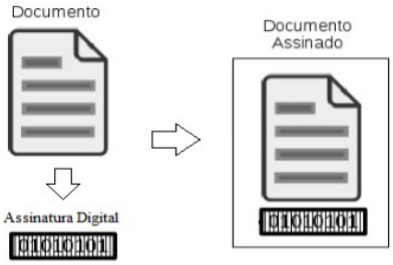
De forma objetiva, a assinatura eletrônica pode ser entendida como qualquer tipo de assinatura realizada por dispositivos eletrônicos.

> Uma assinatura eletrônica representa um conjunto de dados, no formato eletrônico, que é anexado ou logicamente associado a um outro conjunto de dados, também no formato eletrônico, para conferir-lhe autenticidade ou autoria. A assinatura eletrônica, portanto, pode ser obtida por meio de diversos dispositivos ou sistemas, como login/senha, biometria, impostação de *Personal Identification Number* (PIN) etc. (ICP-Brasil, 2015a)

De acordo com a definição mencionada sobre assinatura eletrônica, essa forma de autenticação de documentos já é regulamentada no Brasil desde a Medida Provisória 2.200-2 de 24 de agosto de 2001. No entanto, a partir da Lei n.º 14.063/2020 (BRASIL, 2020), a assinatura eletrônica foi categorizada em três níveis de confiança:

- *Assinatura eletrônica simples*: ocorre quando uma única assinatura é gerada sobre um conteúdo digital disponível. Ela utiliza métodos menos refinados de identificação do signatário, permite identificar o seu signatário e anexa ou associa dados a outros dados em formato eletrônico do signatário. A implementação de uma assinatura simples pode ser vista na Figura 7.
- *Assinatura Eletrônica Avançada*: utiliza certificados que não são emitidos pela ICP-Brasil ou qualquer outro meio reconhecido para comprovação da autoria e integridade de documentos eletrônicos (BRASIL, 2020).
- *Assinatura Eletrônica Qualificada*: utiliza certificado digital emitido pela ICP-Brasil, nos termos da Medida Provisória 2.200-2/2001.

Figura 7 – Assinatura simples em um documento



Fonte: ICP-Brasil (2015a)

Ainda em Brasil (2020), é determinado o uso de diferentes tipos de assinatura eletrônica de acordo com o contexto e o grau de segurança requerido. A assinatura eletrônica simples é admitida em interações de menor impacto com entes públicos e em algumas outras situações específicas. Já a assinatura eletrônica avançada pode ser utilizada nas mesmas situações da simples, além de no registro de atos perante juntas comerciais. Por fim, a assinatura eletrônica qualificada é obrigatória em todas as interações eletrônicas com entes públicos, incluindo as mencionadas anteriormente. Além disso, seu uso é exigido em atos assinados por chefes de Poder, Ministros de Estado, titulares de Poder ou órgão constitucionalmente autônomo de ente federativo, na emissão de notas fiscais eletrônicas (exceto para pessoas físicas ou MEIs), em atos de transferência e registro de bens imóveis, e em outras situações previstas em lei.

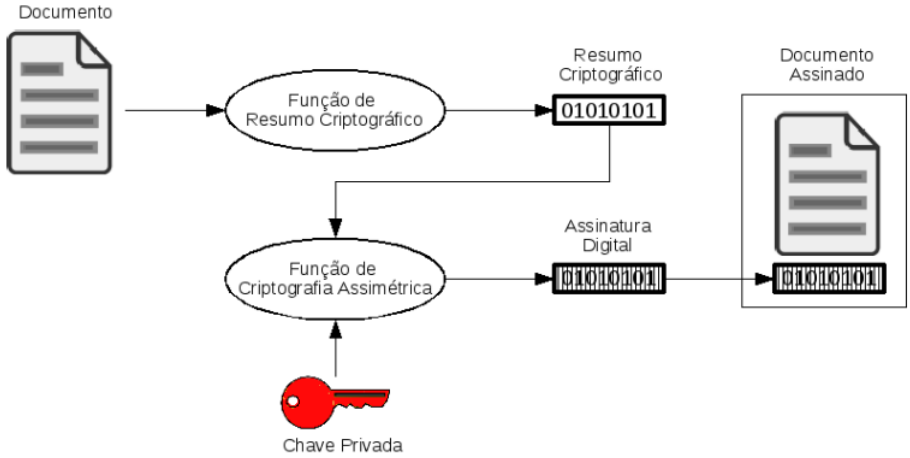
2.5 Assinatura Digital

Com base nas tecnologias já mencionadas, como criptografia, funções de *hash* e certificação, a assinatura digital (assinatura eletrônica avançada e assinatura eletrônica qualificada) se destaca como o método mais robusto de assinatura eletrônica para garantir os requisitos de segurança da informação de uma mensagem: sigilo, autenticação, não repúdio e integridade.

> Uma assinatura digital é um mecanismo de autenticação que permite ao criador de uma mensagem anexar um código que atua como uma assinatura. Normalmente a assinatura é formada pelo *hash* da mensagem e pela encriptação da mensagem com a chave privada do criador. A assinatura garante a origem e a integridade da mensagem. (STALLINGS, 2013)

Sendo assim, reunindo as ideias discutidas é possível diagramar o processo simplificado de uma assinatura digital, que pode ser visto na Figura 8.

Figura 8 – Diagrama simplificado de assinatura digital



Fonte: ICP-Brasil (2015a)

1. O signatário cria um resumo criptográfico do documento eletrônico;
2. O signatário utiliza sua chave privada, que está vinculada a uma chave pública presente em seu certificado digital, para cifrar o resumo criptográfico, gerando assim a assinatura digital;
3. O documento eletrônico e a assinatura digital são vinculados entre si, permitindo sua validação posterior.

A Figura 9 apresenta uma ilustração simplificada do processo de verificação de assinatura digital utilizando o algoritmo RSA.[1], explicado em Rajesh Bondugula (2018):

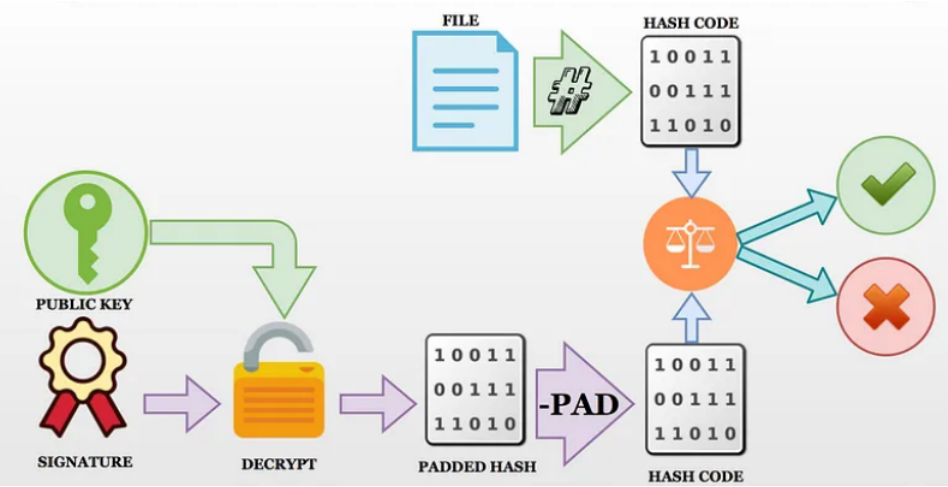
1. O arquivo original passa por um algoritmo de *hash* (exemplo: SHA-256), gerando um código *hash* único (resumo do conteúdo do arquivo);
2. A assinatura digital, que foi criada usando a chave privada do signatário, é descriptografada utilizando a chave pública correspondente. Esse processo revela o *hash* assinado, que foi gerado no momento da assinatura;

[1] O RSA (Rivest-Shamir-Adleman) é um algoritmo de criptografia assimétrica que utiliza um par de chaves: uma pública para criptografar (ou verificar assinaturas) e uma privada para descriptografar (ou assinar). Sua segurança baseia-se na dificuldade de fatorar números grandes (TANENBAUM; FEAMSTER, 2021).

3. Em alguns esquemas de assinatura (exemplo: RSA), a assinatura inclui um preenchimento (*padding*) para segurança. Esse preenchimento é removido, deixando apenas o *hash* original assinado.

4. O *hash* gerado a partir do arquivo e o *hash* obtido da assinatura são comparados. Se os dois valores forem iguais, significa que o documento não foi alterado e a assinatura é válida. Se forem diferentes, significa que o documento foi modificado ou a assinatura não pertence ao signatário;

Figura 9 – Diagrama simplificado de verificação da assinatura digital



Fonte: (Rajesh Bondugula, 2018)

A assinatura digital pode ser formalmente representada por meio de duas funções fundamentais: a função de assinatura S e a função de verificação V. Essas funções abstraem o processo de assinatura digital, tornando-o independente do criptossistema específico utilizado.

A assinatura digital de um documento *doc* utilizando uma chave privada *sk* é definida como:

$$\sigma = S(sk, doc)$$

onde σ representa a assinatura gerada. Esse processo consiste na aplicação de um algoritmo criptográfico que, normalmente, envolve a geração de um hash do documento e sua cifragem com a chave privada do signatário.

A verificação da autenticidade da assinatura digital, por sua vez, é realizada utilizando a chave pública *pk* correspondente à chave privada usada na assinatura. Esse processo é definido pela função:

$$V(pk, doc, \sigma) = \begin{cases} V, & \text{se a assinatura for válida} \\ F, & \text{caso contrário} \end{cases}$$

onde V indica que a assinatura é válida e F indica que a assinatura não pode ser autenticada. O mecanismo de verificação consiste em aplicar a chave pública ao valor assinado σ e compará-lo com o hash do documento original, garantindo sua integridade e autenticidade.

Essa abordagem garante que qualquer modificação no documento após a assinatura resulte em uma verificação falha, tornando a assinatura digital um mecanismo robusto de integridade e autenticação na comunicação digital segura.

2.5.1 Políticas de Assinatura

As políticas de assinatura são um conjunto de critérios técnicos e legais para geração e validação de assinaturas digitais, definindo aspectos como os algoritmos criptográficos permitidos, a necessidade de certificados digitais emitidos por Autoridades Certificadoras confiáveis, entre outros. Além disso, elas garantem a interoperabilidade entre diferentes sistemas e asseguram que os documentos assinados digitalmente sejam aceitos em diversas jurisdições. "Uma assinatura digital é criada pelo signatário de acordo com uma política de assinatura. A validade de uma assinatura digital é avaliada pelo verificador utilizando a mesma política de assinatura usada na criação dessa assinatura digital" (ICP-Brasil, 2021c).

As políticas de assinatura digital estabelecem regras claras sobre como uma assinatura deve ser gerada e verificada, garantindo sua validade jurídica e técnica. A parte que recebe os documentos assinados pode definir quais políticas aceita em seu processo de negócios, assegurando que as assinaturas atendam aos seus requisitos de segurança e conformidade. Dessa forma, o uso de políticas de assinatura proporciona transparência para todas as partes envolvidas, deixando explícitos os critérios que tornam um documento digitalmente assinado válido, como é colocado por ICP-Brasil (2021c).

As políticas de assinatura permitidas na ICP-Brasil são:

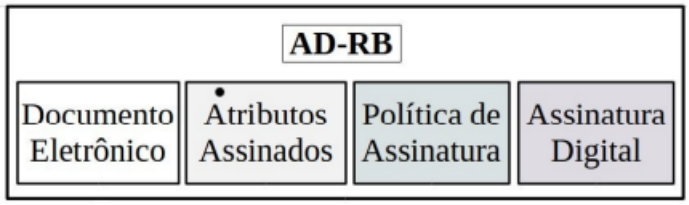
- **Assinatura Digital com Referência Básica (AD-RB)**: Utiliza apenas o certificado digital para validar a assinatura, sem incluir dados adicionais para verificação posterior.
- **Assinatura Digital com Referência do Tempo (AD-RT)**: Inclui um carimbo do tempo, registrando a data e hora em que a assinatura foi realizada.
- **Assinatura Digital com Referências para Validação (AD-RV)**: Adiciona referências externas que possibilitam a verificação posterior, como listas de revogação

de certificados (CRL), que é uma lista de certificados revogados, ou o OCSP (*Online Certificate Status Protocol*), um protocolo que permite verificar em tempo real se um certificado foi revogado, para garantir que o certificado ainda é válido.

- **Assinatura Digital com Referências Completas (AD-RC)**: Incorpora todas as informações necessárias para validação dentro do próprio documento, tornando-o independente de fontes externas.
- **Assinatura Digital com Referências para Arquivamento (AD-RA)**: Inclui todas as informações para validação e preservação de longo prazo, garantindo a validade da assinatura por períodos prolongados.

Para este trabalho, será adotada a AD-RB, utilizando o padrão PAdES (*PDF Advanced Electronic Signatures*). Como é colocado por ICP-Brasil (2015b), a AD-RB é composta por quatro elementos essenciais, que estão na Figura 10: o identificador da política de assinatura, que define as regras utilizadas na criação e verificação da assinatura digital; os dados da assinatura, que incluem informações adicionadas pelo signatário, como o instante de criação; e a sequência de códigos da assinatura, que corresponde ao código criptográfico gerado para garantir a autenticidade, integridade e não repúdio do documento assinado, e o documento eletrônico em si.

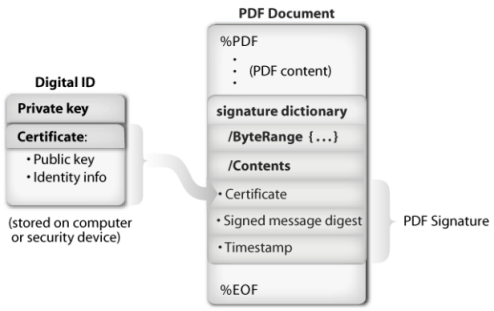
Figura 10 – Assinatura digital com Referência Básica



Fonte: (ICP-Brasil, 2015b).

O PAdES é um padrão para a assinatura de documentos PDF, garantindo autenticidade, integridade e não-repúdio, conforme descrito em ETSI (2009). Ele é compatível com certificados digitais, permite assinatura de longo prazo e, além disso, este padrão possibilita a inclusão de referências internas, mostrado na Figura 11, como o próprio certificado digital e o carimbo de tempo, tornando o documento autossuficiente e facilitando a verificação.

Figura 11 – Inclusão de referências internas no documento PDF



Fonte: (ETSI, 2009).

A política de assinatura adotada neste trabalho segue os princípios estabelecidos em ICP-Brasil (2021a), atendendo aos requisitos mínimos definidos em ICP-Brasil (2021b) e está detalhada no Anexo A.

É importante destacar que esta política possui caráter didático para os fins deste estudo. Como é explicado por ICP-Brasil (2021b), antes de ser efetivamente implementada, primeiramente uma Política de Assinatura no contexto da ICP-Brasil deve ser submetida à CA-Raiz para a obtenção de um identificador. "Identificador da política de assinatura são dados que identificam de forma unívoca uma política de assinatura, compostos por um *Object Identifier* (OID) - ou seja, um identificador - e o resumo criptográfico da política." (ICP-Brasil, 2021c).

No processo de assinatura digital, são incorporados a Política de Assinatura em um formato legível por máquinas, a Lista de Políticas de Assinatura Aprovadas (LPA), que define as diretrizes aceitas para a assinatura, e a assinatura digital da LPA, garantindo sua autenticidade e integridade.

No entanto, como a Política de Assinatura (PA) utilizada neste trabalho é meramente ilustrativa, ela é incluída apenas como um exemplo, utilizando o identificador e a URL fornecidos no Anexo A.

2.5.2 Requisitos para geração e validação de assinaturas digitais

Neste trabalho, a assinatura digital é definida da seguinte forma:

- para **assinatura eletrônica avançada**:
  - esteja associada inequivocamente a um par de chaves criptográficas que permita identificar o signatário;
  - seja produzida por dispositivo seguro de criação de assinatura;
  - esteja vinculada ao documento eletrônico a que diz respeito, de tal modo que qualquer alteração subsequente neste seja plenamente detectável;

- para **assinatura eletrônica qualificada**:
  - tenha todos os itens da assinatura eletrônica avançada;
  - esteja baseada em um certificado ICP-Brasil, válido à época da sua aposição;

Neste contexto, a API desenvolvida no escopo deste trabalho tem como foco a implementação da assinatura eletrônica avançada.

A geração e verificação de uma assinatura digital em um arquivo PDF neste projeto seguem as diretrizes estabelecidas em ICP-Brasil (2015b), com as devidas adaptações. Assim, vale observar os seguintes requisitos:

Para a geração de uma assinatura digital:

- A assinatura deve estar inequivocamente vinculada a uma pessoa ou entidade e ao documento eletrônico correspondente.
- Deve ser realizada dentro do período de vigência do certificado digital.
- Deve estar em conformidade com as restrições do certificado digital.
- Deve utilizar componentes que assegurem:
  - A identificação do documento assinado.
  - A integridade do documento.
  - A identidade do signatário.
  - Os detalhes do certificado empregado.

Para validação de uma assinatura digital:

- Toda assinatura digital deve ser passível de validação.
- Para a validação, são necessários:
  - O documento eletrônico assinado.
  - A assinatura digital.
  - O certificado digital do signatário e sua cadeia de certificação.
- O processo de validação deve:
  - Verificar o estado criptográfico da assinatura.
  - Garantir autenticidade e autoria por meio da decifração da assinatura com a chave pública do certificado digital do signatário.
  - Confirmar a integridade do documento, comparando seus resumos criptográficos para assegurar que não houve alterações desde a assinatura.

2.6 API (*Application Programming Interface*)

Uma API, que pode ser traduzida como Interface de Programação de Aplicações, é um conjunto de definições e protocolos que facilita a comunicação entre diferentes sistemas de software, servindo como uma ponte para que diferentes aplicações troquem informações de forma padronizada e controlada. A API encapsula a lógica de negócio ao abstrair a complexidade das operações internas e oferecer uma interface padronizada para interação. “Uma API oferece uma maneira simples de conectar, integrar e estender sistemas de software”(BIEHL, 2015). Além disso, é possível aplicar difrentes estilos de arquitetura, como por exemplo o REST.

A arquitetura REST é uma forma de estruturar uma API de maneira que ela seja fácil de usar e entender. Em uma API REST, a ideia principal é que o sistema seja organizado em torno de "recursos", que são as informações ou entidades que a API manipula. Esses recursos podem ser, por exemplo, um usuário, um documento ou um produto. Cada recurso é acessado por meio de um *endpoint*, que pode ser entendido como uma URL específica que representa uma operação ou recurso disponível na API. Por meio desses *endpoints*, usuários ou programas podem enviar requisições para interagir com os dados da API

Como é colocado por Biehl (2015), as APIs REST são projetadas para realizar operações fundamentais de gerenciamento de dados, conhecidas como CRUD (criar, ler, atualizar e excluir). Essas operações podem ser associadas diretamente aos métodos HTTP. A criação de novos recursos é feita com os métodos *POST* ou *PUT*, enquanto a leitura de informações é feita com o método *GET*. Para atualizar um recurso existente, utiliza-se o método *PUT*, e a exclusão de um recurso é feita com o método *DELETE*.

Por fim, Biehl (2015) também destaca vantagens desse tipo de estilo de API que será importante para o desenvolvimento do projeto: as APIs REST oferecem benefícios como escalabilidade e resiliência, pois são sem estado e independentes, permitindo a adição de servidores para expandir a capacidade do sistema. Elas também se beneficiam do cache integrado da infraestrutura HTTP, melhorando o desempenho sem necessidade de alterações adicionais. REST suporta diferentes formatos de conteúdo e permite a negociação entre cliente e servidor. A padronização dos métodos facilita a exploração e uso de APIs, tornando-as intuitivas e acessíveis. Além disso, o uso correto de métodos e códigos de status HTTP assegura a visibilidade e monitoramento das interações entre componentes.

2.7 *Web App*

Uma *Web App*, ou Aplicação Web, é um software acessado por meio de um navegador de internet, que permite ao usuário interagir e realizar operações em um ambiente

digital. Diferente de um site estático, uma aplicação web é projetada para responder a ações do usuário, processar dados e oferecer funcionalidades específicas, como formulários de envio, painéis interativos, sistemas de login, entre outros. “As aplicações Web têm uma arquitetura de cliente-servidor. Seu código é dividido em dois componentes: scripts do lado do cliente e scripts do lado do servidor.” (AWS, 2020). No lado do cliente, o script é responsável pela interface e interação com o usuário, como botões e formulários. Quando o usuário interage com a aplicação, como clicar em um botão, o navegador processa essa ação. No lado do servidor, os scripts processam as solicitações do cliente, como salvar ou recuperar dados de um banco de dados, e retornam uma resposta. Em alguns casos, o servidor envia a página completa já processada de volta para o cliente, processo conhecido como renderização no lado do servidor.

3 Metodologia

O presente capítulo trata da metodologia adotada neste trabalho, abordando a classificação da pesquisa, o fluxo de atividades para a elaboração do TCC, a metodologia de pesquisa do referencial teórico e a metodologia de desenvolvimento do sistema proposto. Além disso, é discutida a gerência de configuração e evolução de software para garantir a qualidade e manutenibilidade do sistema ao longo do tempo.

> O método é o conjunto das atividades sistemáticas e racionais que, com maior segurança e economia, permite alcançar o objetivo - conhecimentos válidos e verdadeiros -, traçando o caminho a ser seguido, detectando erros e auxiliando as decisões do cientista. (MORESI, 2003).

3.1 Classificação

A pesquisa apresentada neste trabalho é classificada quanto a sua natureza, sua abordagem e seus fins:

- Quanto à natureza, é uma pesquisa aplicada, pois visa desenvolver uma solução para um problema específico;
- Quanto à abordagem, é uma pesquisa qualitativa privilegiando a riqueza de detalhes e a compreensão contextual. Através de ferramentas como observações e análises de documentos;
- Quanto aos fins, é exploratório, buscando explorar a implementação de técnicas de criptografia, certificação e assinatura eletrônica no contexto do desenvolvimento do software de assinatura digital. O objetivo é entender as possibilidades e desafios relacionados a essas técnicas, bem como identificar as melhores práticas para sua implementação.

3.2 Metodologia para Pesquisa do Referencial Teórico

Esta fase construtiva, conforme Moresi (2003), envolve a elaboração de um plano de pesquisa e a realização efetiva da pesquisa, sendo desenvolvida em duas etapas distintas. A exploração dos conteúdos nessas etapas foi conduzida por meio de pesquisa em livros, artigos acadêmicos, sites governamentais e outros recursos relevantes que abordam temas relacionados à assinatura digital.

Na primeira etapa, foi realizado um estudo sobre o funcionamento da assinatura digital. O objetivo era decompor o processo em partes que pudessem ser explicadas gradualmente, permitindo assim a compreensão detalhada de cada fase, desde a geração das chaves até a verificação da integridade do documento assinado.

A segunda etapa teve como objetivo compreender a legislação de chaves públicas e assinaturas eletrônicas no Brasil. Isso envolveu a análise de leis, medidas provisórias e regulamentações que regem o uso e a validade das assinaturas digitais no país. Essa análise contribuiu para uma compreensão abrangente do contexto legal em que o software de assinatura digital será desenvolvido, garantindo sua conformidade com as normas vigentes.

Essa abordagem metodológica permitiu não apenas a compreensão dos aspectos técnicos da assinatura digital, mas também a consideração dos aspectos legais e regulatórios relevantes para o contexto brasileiro.

3.3 Metodologia de Desenvolvimento

As metodologias de desenvolvimento constituem um conjunto de técnicas e métodos organizacionais empregados para conceber e implementar soluções. Bourque e Fairley (2014) explica que há uma variedade de métodos disponíveis e é crucial que o engenheiro de software selecione métodos apropriados para a tarefa de desenvolvimento de software em questão. Essa escolha pode impactar significativamente o sucesso do projeto de software.

A metodologia para o desenvolvimento destre trabalho é uma mescla dos chamados métodos ágeis.

> Os métodos ágeis são considerados métodos leves, na medida em que se caracterizam por ciclos de desenvolvimento curtos e iterativos, equipes auto-organizáveis, concepções mais simples, refatorização do código, desenvolvimento orientado para os testes, envolvimento frequente do cliente e ênfase na criação de um produto de trabalho demonstrável em cada ciclo de desenvolvimento (BOURQUE; FAIRLEY, 2014).

No *Scrum*, uma lista de Itens do Backlog do Produto (PBI) é criada, na qual as tarefas são identificadas, definidas, priorizadas e estimadas. Cada incremento do software é testado e lançado, resultando em uma versão funcional a cada ciclo. Os artefatos utilizados do *Scrum* são:

- ***Backlog* do Produto**: lista de funcionalidades e requisitos de implementação do projeto;
- ***Sprint***: intervalo de tempo com duração de duas semanas para desenvolver itens selecionados do *Backlog*;

- **Reunião de Revisão da *Sprint***: Reunião para mostrar ao *stakeholder*, que no caso deste projeto foi o professor orientador, o que foi feito em relação a entrega daquela *Sprint*;

O *Kanban* é uma abordagem de gestão visual que visa guiar cada tarefa através de um fluxo de trabalho predefinido. Ele não é apenas um sistema visual de gestão de trabalho, mas também uma metodologia que enfatiza a visualização do trabalho em progresso e a limitação da quantidade de trabalho em execução simultaneamente. Os artefatos utilizados dessa metodologia são:

- **Visualização do Fluxo de Trabalho**: Garantir que todas as etapas do trabalho estejam claras, pois o trabalho invisível pode representar riscos para o projeto. O quadro Kanban oferece uma representação clara do fluxo de trabalho.
- **Implementação de Feedback**: envolve a análise contínua do desempenho do processo e a adaptação com base nas observações. É importante coletar feedback regularmente dos membros da equipe e dos stakeholders;

O XP, ou *Extreme Programming*, é uma metodologia ágil de desenvolvimento de software que enfatiza a entrega rápida e frequente de software funcional de alta qualidade. As práticas-chave do XP utilizadas no projeto são:

- **Integração Contínua**: cada nova parte desenvolvida é integrada ao código de forma a assegurar que as mudanças não causem interferências no que já foi desenvolvido. Cada integração é verificada automaticamente por meio de builds e testes automatizados, garantindo que o código seja funcional e compatível com o restante do projeto;
- **Refatoração**: processo de melhorar o código existente sem alterar seu comportamento externo, tornando-o mais limpo, compreensível e fácil de mante;

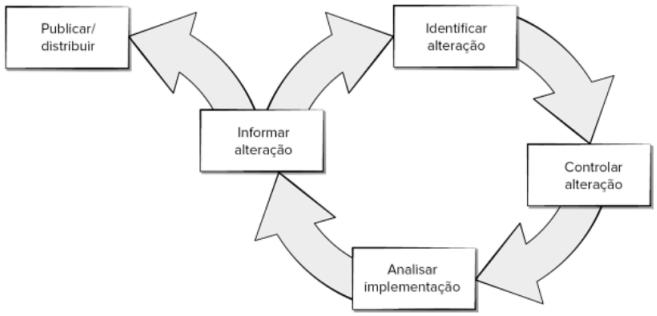
3.3.1 Gerência de Configuração e Evolução de Software

Ainda nas palavras de Bourque e Fairley (2014), a Gerência de Configuração e Evolução de Software (GCES) é uma atividade abrangente que ocorre durante todo o ciclo de vida de um software. Ela é responsável por gerenciar e controlar a evolução do software, incluindo o controle de versões e solicitações de mudanças. Isso permite que todas as partes envolvidas na criação e manutenção do software tenham acesso ao histórico de modificações, o que fornece informações para compreender o sistema na sua forma atual e em suas formas anteriores.

O fluxo de GCES pode ser visto na Figura 12 abaixo. Pressman e Maxim (2021) descreve esse fluxo da seguinte forma: dado que as mudanças podem acontecer a qualquer

momento, as atividades de GCES são projetadas para (1) identificar a mudança, (2) controlá-la, (3) garantir que seja implementada corretamente e (4) comunicar as mudanças para outros envolvidos.

Figura 12 – Fluxo de trabalho de GCES



Fonte: Pressman e Maxim (2021)

.

Tendo em vista este fluxo de trabalho para gerenciar o projeto, é necessário utilizar ferramentas que permitam sua aplicação utilizando os artefatos das metodologias de desenvolvimento mencionados acima. Um aspecto fundamental disso é o controle de versão, que "[...]combina procedimentos e ferramentas para gerenciar diferentes versões dos objetos de conguração criados durante o processo de software" (PRESSMAN; MAXIM, 2021). Além disso, é necessário ter um repositório centralizado para armazenar e compartilhar o código-fonte e outros artefatos do projeto, que sirva como um ponto central onde é possível controlar as diferentes versões do software.

Para o controle de versões será utilizado o Git: "O Git é um sistema de controle de versão distribuído, gratuito e de código aberto, projetado para lidar com tudo, de projetos pequenos a muito grandes, com rapidez e eficiência" (Git, 2024).

O repositório do projeto será hospedado no GitHub, que é uma plataforma de hospedagem de repositórios Git na nuvem, que oferece recursos adicionais como controle de acesso, gerenciamento de problemas (issues), integração contínua e revisão de código.

Essas ferramentas desempenham um papel crucial na aplicação eficaz das metodologias ágeis. No Scrum, o Git permite o versionamento do código, enquanto o GitHub ajuda a organizar o Backlog do Produto e acompanhar o progresso das sprints. No Kanban, o Git e o GitHub proporcionam uma visão clara do fluxo de trabalho, representando os itens do backlog como issues, além de facilitar a integração contínua e a implementação de feedbacks. No XP, o Git é essencial para práticas como Integração Contínua (CI), enquanto o GitHub suporta revisão de código e melhorias contínuas por meio de refatorações.

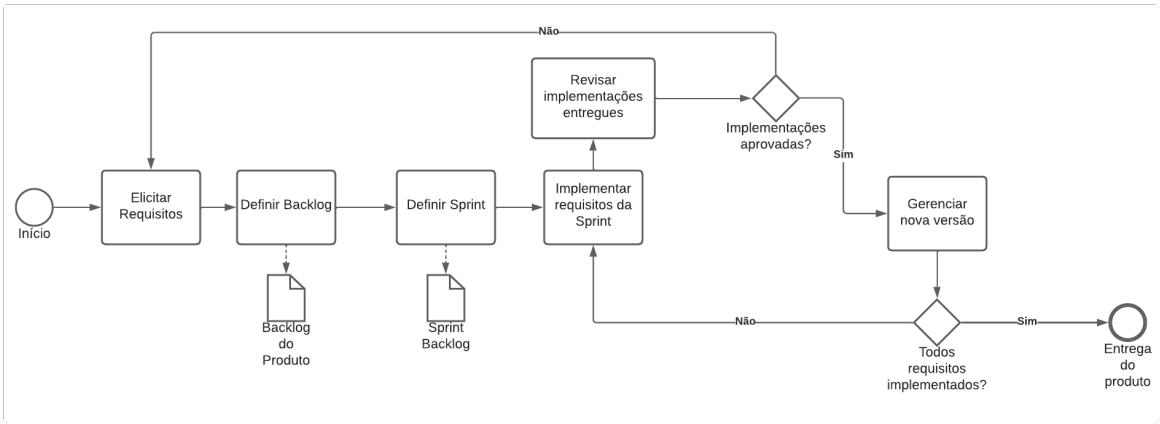
Neste projeto, foi utilizado uma Organização do GitHub, dividindo o trabalho em

três repositórios, o que garantiu um melhor controle das mudanças e uma gestão ágil do desenvolvimento de software.

3.3.2 Fluxo de Desenvolvimento

Na Figura 13 estão representadas as etapas do fluxo de desenvolvimento. Com a adoção de uma metodologia híbrida para o projeto, observa-se que o fluxo se caracteriza por ciclos de desenvolvimento curtos e iterativos, além de refatorização do código, enfocando práticas de Scrum, Kanban e XP.

Figura 13 – Fluxo de Desenvolvimento do Projeto



Fonte: Autor.

A descrição das estapas é a seguinte:

- **Elicitar requisitos**: Os requisitos do sistema são identificados por meio de histórias de usuários que descrevem as funcionalidades desejadas do ponto de vista do usuário, ajudando a entender as necessidades e expectativas em relação ao produto final;
- **Definir backlog**: Gerar o backlog do produto, que é uma lista priorizada de todos os requisitos elicitados na etapa anterior. O artefato gerado serve como guia para o desenvolvimento;
- **Definir sprint**: Um conjunto de itens do backlog é selecionado para ser implementado durante a sprint;
- **Implementar requisitos da sprint**: implementar as funcionalidades e requisitos definidos no backlog da sprint. O objeitov é desenvolver e testar o código, seguindo as práticas estabelecidas pela metodologia adotada;
- **Revisar implementações**: é realizada uma revisão para garantir que as funcionalidades atendam aos requisitos e expectativas estabelecidos. Se as implementações

forem aprovadas, novos requisitos são adicionados para implementação. Caso contrário, os requisitos em questão são revistos e ajustados para serem implementados de acordo com as expectativas.

- **Gerenciar nova versão**: Após a revisão e aprovação das implementações, uma nova versão do software é gerada e gerenciada. Isso inclui o controle de versão do código-fonte, registro das alterações realizadas, gerenciamento de branches e releases, garantindo assim a integridade e evolução do produto ao longo do tempo.

4 Desenvolvimento do Software

Este capítulo apresenta o processo de desenvolvimento do software proposto, detalhando suas principais etapas e decisões técnicas. Inicialmente, são descritas as histórias de usuário e os requisitos. Em seguida, as tecnologias utilizadas na implementação da API UnBSign, da aplicação web e da API PKI são apresentadas.

Além disso, este capítulo aborda a arquitetura do sistema, destacando a estrutura da API REST, da aplicação web e da API PKI, explicando seus componentes e interações.

4.1 Épicos, Histórias de Usuário e Requisitos

Nesta seção, é apresentado a organização dos épicos, histórias de usuários e requisitos que guiam o desenvolvimento da API de assinatura digital, chamada a partir daqui de **API UnBSign**. Os épicos representam funcionalidades de alto nível, enquanto as histórias de usuários detalham as necessidades e expectativas dos usuários em relação ao sistema. Com base nesses elementos, é definido os requisitos funcionais, que especificam de forma clara e objetiva o comportamento esperado da API e da *Web App*. Essa abordagem estruturada visa garantir que as funcionalidades atendam às demandas dos usuários e estejam alinhadas com os objetivos do projeto.

Como será explicado posteriormente, a PKI será implementada utilizando uma abordagem simplificada, por meio de diretórios e arquivos, sem a complexidade de uma infraestrutura distribuída e de alto nível. Nesse modelo, a gestão das chaves e certificados será realizada de maneira direta e acessível, dispensando requisitos avançados ou configurações complexas, adequando-se à finalidade e aos objetivos do sistema.

4.1.1 Épicos e Histórias de Usuário

Os épicos e histórias de usuários são explicados em Bourque e Fairley (2014). Os épicos são descrições de alto nível que englobam grandes funcionalidades ou objetivos do sistema, servindo como um ponto de partida para decomposição em histórias de usuários mais específicas. Eles representam as principais áreas de valor do projeto, como autenticação, assinatura digital de documentos, validação de documentos assinados e gestão de usuários. No contexto deste projeto, os épicos permitem organizar e priorizar o desenvolvimento das funcionalidades, garantindo que cada aspecto essencial da API de assinatura digital seja abordado de forma integrada e eficiente. Na Tabela 1 está os épicos do projeto.

Já as histórias de usuários são descrições curtas e centradas no usuário que detalham necessidades específicas relacionadas aos épicos, conectando os objetivos gerais às

Tabela 1 – Épicos

ID	Descrição
E1	Autenticação e Autorização: Garantir que apenas usuários autenticados e autorizados possam acessar as funcionalidades da API
E2	Upload e Visualização de Documentos: Permitir que o usuário envie um PDF e visualize o conteúdo antes de prosseguir com a assinatura
E3	Assinatura Digital: Permitir que o usuário assine digitalmente o documento
E4	Validação de Documentos Assinados: Validar a integridade e autenticidade de um documento já assinado

Fonte: Autor.

funcionalidades práticas do sistema. Cada história reflete o que um usuário deseja alcançar, como visualizar documentos enviados ou assinar um PDF com um certificado digital, e ajuda a guiar a implementação técnica e a priorização das entregas. Na Tabela 2 apresentada, as histórias de usuários são agrupadas por épicos, oferecendo uma visão clara da relação entre as metas amplas do sistema e os requisitos detalhados que sustentam sua construção.

Tabela 2 – Histórias de Usuário

Épico	ID	História de Usuário	Critérios de aceitação
E1	US1	Como usuário, desejo me autenticar no sistema	A autenticação deve ser realizada com email e senha.
E2	US2	Como usuário, desejo enviar um arquivo PDF	Sistema aceitar apenas arquivos no formato PDF.
	US3	Como usuário, desejo visualizar o PDF enviado	Visualizar as páginas do PDF enviado antes de assinar.
E3	US4	Como usuário, desejo selecionar o tipo de certificado digital utilizado para assinar o PDF	Opções: Certificado Digital da Instituição ou Certificado próprio ICP-Brasil.
	US5	Como usuário, desejo escolher a posição do carimbo da assinatura no PDF	O sistema deve permitir ao usuário selecionar as coordenadas do carimbo da assinatura no documento. O carimbo deve conter informações relevantes (Nome do assinante, data e hora).

Épico	ID	História de Usuário	Critérios de aceitação
	US6	Como usuário, desejo assinar o PDF com certificado escolhido	O sistema deve realizar a assinatura digital do PDF utilizando o padrão PAdES. O PDF deve ser retornado ao usuário assinado para que possa ser feito o download.
E4	US7	Como usuário, desejo enviar um PDF para validação	O retorno deve indicar se o documento é válido e detalhes sobre a assinatura.
	US8	Como usuário, desejo visualizar os detalhes da assinatura digital no documento	O sistema deve retornar informações como o certificado usado, data/hora e validade da assinatura.

Fonte: Autor.

A partir das histórias de usuários, foram elicitados os requisitos funcionais para a API e a *Web App*, estabelecendo de forma clara as funcionalidades necessárias para atender às demandas identificadas. Essa etapa garante que cada história seja traduzida em especificações técnicas, divididas entre a API e a interface do usuário, promovendo uma implementação coerente e alinhada com os objetivos do sistema.

4.1.2 Requisitos

Os **requisitos** podem ser definidos, no contexto de desenvolvimento de software, como as funcionalidades, capacidades e restrições de um sistema de software. O principal objetivo dos requisitos é orientar o desenvolvimento para garantir que o software atenda às necessidades e expectativas idealizadas sobre o produto final. Os requisitos funcionais são sobre as funções que o software deve executar. "Um requisito funcional também pode ser descrito como aquele para o qual um conjunto finito de etapas de teste pode ser escrito para validar seu comportamento" (BOURQUE; FAIRLEY, 2014).

A elicitação de requisitos consiste em identificar as fontes de requisitos de software e determinar como o engenheiro de software pode obtê-los. Trata-se de uma etapa crucial para desenvolver uma compreensão clara do problema que o software deve resolver, como destacado por Bourque e Fairley (2014). Esse processo pode ser realizado por meio de diferentes técnicas, escolhidas com base nas fontes de requisitos e nos objetivos almejados. Neste trabalho, os requisitos foram elicitados a partir das histórias de usuários e pela aplicação da técnica de *Benchmarking*, que permitiu comparar soluções existentes para garantir um alinhamento eficiente às necessidades do sistema proposto.

A técnica de *Benchmarking* envolve a análise dos processos executados por empresas concorrentes ou ideias similares, especialmente aqueles que apresentam desafios ou dificuldades semelhantes aos enfrentados pela empresa em questão. No contexto deste trabalho, essa abordagem é valiosa tanto para o levantamento de requisitos quanto para a concepção da interface do sistema. Ao examinar como outras organizações lidam com determinados processos ou problemas, é possível identificar melhores práticas, soluções inovadoras e áreas de melhoria que podem inspirar e informar o desenvolvimento do sistema. Essa análise comparativa não apenas ajuda a compreender o estado atual do mercado e das tecnologias disponíveis, mas também pode fornecer insights cruciais para a diferenciação e aprimoramento do próprio sistema.

Após a elicitação, os requisitos passam pela etapa de priorização, que determina quais devem ser implementados primeiro com base em critérios como valor para o negócio, urgência, viabilidade técnica e impacto no usuário final. A técnica utilizada para isso é a **MoSCoW**. Conforme explicado por Wiegers e Beatty (2013), as letras maiúsculas representam as seguintes prioridades:

- ***Must*** (Deve): O requisito deve ser atendido para que a solução seja considerada um sucesso.
- ***Should*** (Deveria): O requisito é importante e deve ser incluído na solução, se possível, mas não é obrigatório para o sucesso.
- ***Could*** (Poderia): É um recurso desejável, mas que pode ser adiado ou eliminado. Implemente-o somente se o tempo e os recursos permitirem.
- ***Won't*** (Não será): Indica um requisito que não será implementado no momento, mas que poderá ser incluído em uma versão futura.

Os requisitos funcionais elicitados para a *Web App* e API a partir das histórias de usuário e benchmarking podem ser vistos na Tabela 3 e na Tabela 4. Os identificadores são RFWx para a *Web App* e RFAx para a API.

Tabela 3 – Requisitos Funcionais para *Web App*

ID	Requisito	Prioridade
RFW01	A *Web App* deve fornecer uma tela de cadastro onde o usuário possa inserir as seguintes informações: nome completo, apelido, data de nascimento e senha	*Must*
RFW02	A *Web App* deve permitir que o usuário faça login utilizando seu apelido e senha	*Must*

ID	Requisito	Prioridade
RFW03	O *backend*, ao autenticar usuário, deve retornar um token JWT (JSON *Web Token*) válido	*Must*
RFW04	A *Web App* deve armazenar o token JWT no *cookie* do navegador.	*Must*
RFW05	A *Web App* deve incluir esse token em todas as requisições subsequentes, no cabeçalho, para acessar rotas protegidas da API	*Must*
RFW06	A *Web App* deve disponibilizar um campo de upload para que o usuário envie um arquivo PDF	*Must*
RFW07	O campo de upload deve ser validado para garantir que apenas arquivos com a extensão .pdf sejam aceitos.	*Should*
RFW08	A *Web App* deve exibir uma pré-visualização do PDF logo após o upload bem-sucedido.	*Should*
RFW09	A *Web App* deve exibir o PDF carregado de forma interativa, permitindo que o usuário visualize as páginas do documento e identifique o local adequado para a assinatura	*Should*
RFW10	O usuário deve ser capaz de selecionar a posição exata do carimbo dentro do PDF, por meio de uma interface de arraste, clicando e arrastando o carimbo para a posição desejada no PDF carregado para pré-visualização	*Should*
RFW11	Disponibilizar botão de “Assinar” ao carregar um PDF.	*Must*
RFW12	Ao clicar em "Assinar", envia o arquivo PDF, a posição do carimbo e o token JWT para a API.	*Must*
RFW13	Exibir o PDF assinado quando o processo for concluído e permitir download.	*Could*
RFW14	A *Web App* deve permitir que o usuário envie um documento PDF assinado para validação	*Should*
RFW15	A *Web App* deve ter um botão "Validar Documento", que envia o arquivo PDF assinado para a API	*Should*
RFW16	A *Web App* deve exibir uma mensagem de sucesso ou erro com base no resultado da validação.	*Should*
RFW17	Ao validar, a *Web App* deve apresentar as informações da assinatura em caso de uma assinatura válida no documento.	*Could*

Tabela 4 – Requisitos da API

ID	**Requisito**	**Prioridade**
RFA01	A API permite a autenticação do usuário por meio de um token JWT válido, que será enviado no cabeçalho das requisições subsequentes.	*Should*
RFA02	A API valida o token JWT nas rotas protegidas para garantir que o usuário está autenticado.	*Must*
RFA03	A API expõe um *endpoint* para requisição POST.	*Must*
RFA04	A API recebe um arquivo PDF através de uma requisição POST no *endpoint*.	*Must*
RFA05	A API recebe a posição do carimbo e o número da página do arquivo PDF através de uma requisição POST no *endpoint*.	*Must*
RFA06	A API armazena certificado digital ICP-Brasil institucional	*Won't*
RFA07	A API realiza a assinatura digital do PDF utilizando a assinatura eletrônica qualificada utilizando certificado digital ICP-Brasil institucional	*Won't*
RFA08	A API realiza a assinatura digital do PDF utilizando a assinatura eletrônica qualificada utilizando certificado digital ICP-Brasil fornecido pelo usuário	*Won't*
RFA09	A assinatura digital segue o padrão PAdES, garantindo a conformidade com as regulamentações de assinaturas digitais.	*Should*
RFA10	A API garante que a assinatura digital seja gerada de forma a manter a integridade do arquivo original, assegurando que qualquer alteração posterior ao processo de assinatura invalidará a assinatura.	*Must*
RFA11	A API retorna o PDF assinado na mesma requisição.	*Must*
RFA12	A lógica de assinatura utiliza um mecanismo de *hash* para criar um resumo do documento que será assinado, garantindo que o conteúdo não possa ser alterado sem invalidar a assinatura.	*Must*
RFA13	A API inclui um sistema de verificação que confira a validade do certificado usado para a assinatura, assegurando que o certificado não esteja expirado ou revogado antes de gerar a assinatura.	*Could*

ID	Requisito	Prioridade
RFA14	A API suporta a inclusão de metadados adicionais na assinatura, como a hora exata da assinatura e o identificador do processo, para rastreabilidade.	*Could*
RFA15	A lógica de assinatura é implementada de forma modular para permitir futuras atualizações ou suporte a diferentes algoritmos de assinatura sem impactar o funcionamento da API.	*Should*
RFA16	A API tem um mecanismo interno de auditoria que registre cada etapa do processo de assinatura, incluindo a entrada de dados, a geração de *hashes*, a assinatura propriamente dita e a finalização do arquivo assinado.	*Won't*
RFA17	A API realiza um processo de verificação automática de integridade, comparando o documento assinado com uma versão de teste para garantir que não haja diferença após a assinatura.	*Could*
RFA18	A API responde em um formato JSON padronizado para facilitar a integração com sistemas externos e assegurar que mensagens de erro e sucesso sejam de fácil compreensão.	*Should*
RFA19	A API implementa logs de auditoria em tempo real para monitorar e alertar sobre possíveis anomalias ou tentativas de acesso não autorizado.	*Won't*
RFA20	A API expõe um *endpoint*, onde um documento assinado pode ser enviado para validação da assinatura digital.	*Must*
RFA21	A API inclui documentação completa e interativa, como Swagger/OpenAPI, para facilitar a integração de desenvolvedores.	*Must*
RFA22	A API tem suporte a testes de integração e *endpoints* simulados para que desenvolvedores possam testar a assinatura e validação de documentos sem afetar o ambiente de produção.	*Could*
RFA23	A API valida a assinatura do documento e verifica sua autenticidade.	*Must*
RFA24	A API retorna os detalhes da assinatura digital, incluindo o nome do assinante, o certificado utilizado e o status da assinatura (válido ou inválido).	*Must*

ID	Requisito	Prioridade
RFA25	A API retorna mensagens de erro claras caso haja algum problema com a assinatura ou a validação do documento (ex: erro de formato, erro de assinatura, token inválido).	*Should*

Os requisitos funcionais identificados e priorizados para este projeto podem e serão incrementados à medida que a proposta evoluir. Este processo de refinamento está previsto no fluxo de desenvolvimento mostrado na Figura 13.

4.2 Aplicação da Metodologia de Desenvolvimento

O projeto foi desenvolvido utilizando a abordagem ágil, estruturado em cinco sprints para atender aos requisitos da API de Assinatura Digital e do *Web App*, garantindo entregas incrementais e funcionais. O planejamento de cada sprint foi guiado pelo *sprint backlog*, priorizando as tarefas conforme os objetivos de cada iteração. A gestão das atividades e o monitoramento do progresso foram realizados por meio da ferramenta de projetos do GitHub, utilizando *issues* e *boards*.

O ciclo de desenvolvimento dividido em cinco sprints que tiveram como objetivo:

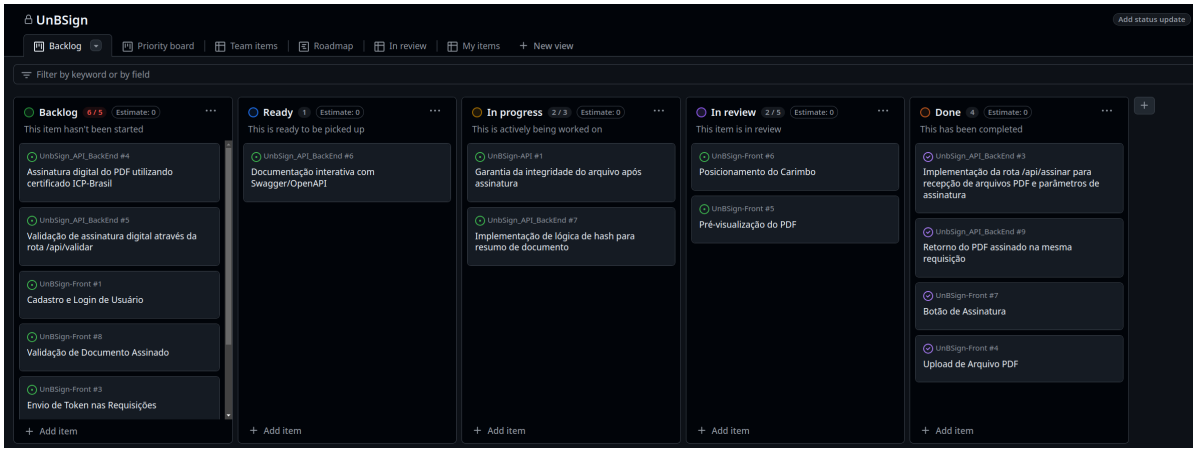
- **Sprint 1:** Implementação do upload de arquivos e lógica de assinatura digital.
- **Sprint 2:** Criação de carimbos digitais com posicionamento e autenticação de usuários via JWT.
- **Sprint 3:** Implementação da interface e rota para validação de documentos.
- **Sprint 4:** Realização de testes funcionais e refinamento de controles de segurança.
- **Sprint 5:** Documentação detalhada da API e ajustes finais na interface.

O quadro Kanban foi dividido em cinco colunas:

- ***Backlog*:** Tarefas ainda não iniciadas
- ***Ready*:** Atividades prontas para execução
- ***In Progress:*** Tarefas em andamento
- ***In Review*:** Tarefas concluídas e em revisão
- ***Done*:** Tarefas finalizadas

O quadro pode ser visto na Figura 14. Essa organização visual e incremental permitiu acompanhar o progresso e garantir entregas de qualidade ao longo do desenvolvimento.

Figura 14 – Quadro Kanban



Fonte: Autor.

4.3 Tecnologias

4.3.1 Tecnologias da API UnBSign

O desenvolvimento da API foi realizado em Java, devido às soluções que oferece para a construção de aplicações escaláveis. A escolha da linguagem foi complementada pelo uso de frameworks modernos e bibliotecas especializadas, que facilitaram a implementação de funcionalidades essenciais. A seguir, são apresentados os principais aspectos técnicos e as ferramentas utilizadas na criação desta solução.

- Java como linguagem de desenvolvimento
  - Amplo ecossistema de frameworks e bibliotecas que melhoram a produção de aplicações complexas, desde a gestão de banco de dados até a disponibilização de rotas. Isso acelera o desenvolvimento, reduz a quantidade de código necessária;
  - Segurança e suporte multiplataforma: possui recursos integrados para garantir a segurança de aplicações, como controle de acesso, verificação de código e suporte a criptografia. Além disso, a portabilidade da JVM permite que aplicações Java sejam executadas de maneira consistente em diferentes sistemas operacionais, tornando-a uma escolha ideal para ambientes corporativos que exigem alta segurança e flexibilidade.
- Spring Framework e Spring Boot <https://spring.io/projects/spring-boot>

  - plataforma popular para desenvolvimento de aplicações em Java, conhecida por sua flexibilidade e arquitetura orientada a aspectos, que permite a criação de sistemas modulares de alta qualidade.
  - uma extensão do Spring que simplifica a configuração e inicialização de projetos, eliminando a complexidade da configuração manual e acelerando o desenvolvimento de APIs REST.
  - Benefícios do Spring Boot: *starters* para configuração automática de dependências;

- Bibliotecas adicionais
  - iTextPDF (<https://itextpdf.com/>): utilizada para criação e manipulação de documentos PDF, permitindo gerar PDFs dinâmicos e personalizados de maneira prática e eficiente.
  - BouncyCastle (<https://www.bouncycastle.org/>): biblioteca de segurança que fornece suporte para criptografia, assinaturas digitais e outras operações essenciais para garantir a proteção e integridade dos dados na API.

- Segurança com JWT
  - JSON Web Tokens (JWT): método de autenticação e controle de acesso que garante que apenas usuários autenticados possam acessar *endpoints* protegidos.
  - Vantagens: tokens autossuficientes que mantêm a identidade do usuário de forma segura, sem a necessidade de armazenar sessões no servidor.
  - Proporciona escalabilidade e integração eficiente entre diferentes partes do sistema, assegurando a confidencialidade e integridade das transações.

4.3.2 Tecnologias da *Web App*

A *Web App* foi projetada como uma demonstração prática da API de Assinatura Digital, ilustrando a implementação de suas rotas e abordando aspectos relacionados à segurança no acesso e na manipulação de dados. Dividida entre a arquitetura do cliente e do servidor, a aplicação oferece uma visão clara de como os dois componentes interagem para criar uma experiência coesa e funcional. A arquitetura do servidor foi desenvolvida utilizando FastAPI, enquanto a arquitetura do cliente utiliza tecnologias como HTML, CSS, e JavaScript para uma interface simples, mas que cumpre com o que proposto.

- FastAPI (<https://fastapi.tiangolo.com/>) para o *backend*: é um framework moderno e de alto desempenho para construção de APIs web, baseado em Python. Ele oferece suporte integrado a validações de dados, geração automática de documentação (OpenAPI) e integração simples com tecnologias de autenticação, o que o torna

ideal para aplicações que precisam demonstrar aspectos de segurança e usabilidade de rotas.

- HTML, CSS e JavaScript para o *front-end*: A escolha de HTML e CSS garante uma estrutura visual clara e estilizada, ideal para a criação de interfaces intuitivas. O JavaScript permite a implementação de funcionalidades interativas no lado do cliente, como validações em tempo real e requisições assíncronas à API via Fetch API. O uso dessas tecnologias fundamenta-se em sua simplicidade e ampla compatibilidade com navegadores, atendendo ao objetivo de demonstrar a funcionalidade da API sem complexidades adicionais.

A aplicação foi desenvolvida para demonstrar, de maneira prática, como a API gerencia autenticação, controle de acesso e validação de assinaturas digitais. Essa abordagem proporciona um ambiente educacional útil tanto para desenvolvedores quanto para avaliadores que desejam compreender a aplicação e a segurança da API em diferentes cenários.

4.3.3 Tecnologias da API PKI

A implementação da Infraestrutura de Chave Pública (PKI) foi simplificada utilizando o OpenSSL, uma das ferramentas mais confiáveis e amplamente adotadas para a criação, gerenciamento e validação de certificados digitais. O OpenSSL fornece uma ampla gama de recursos de criptografia, incluindo a geração de chaves públicas e privadas, a criação de solicitações de assinatura de certificado (CSR), e a assinatura de certificados, tornando-o ideal para a construção de soluções de segurança.

A integração do OpenSSL com o Spring Boot, com sua integração com bibliotecas de terceiros, possibilita a criação de uma API de fácil implementação, que atende às necessidades de segurança de uma infraestrutura PKI, como autenticação, controle de acesso e validação de assinaturas digitais.

4.4 Arquitetura

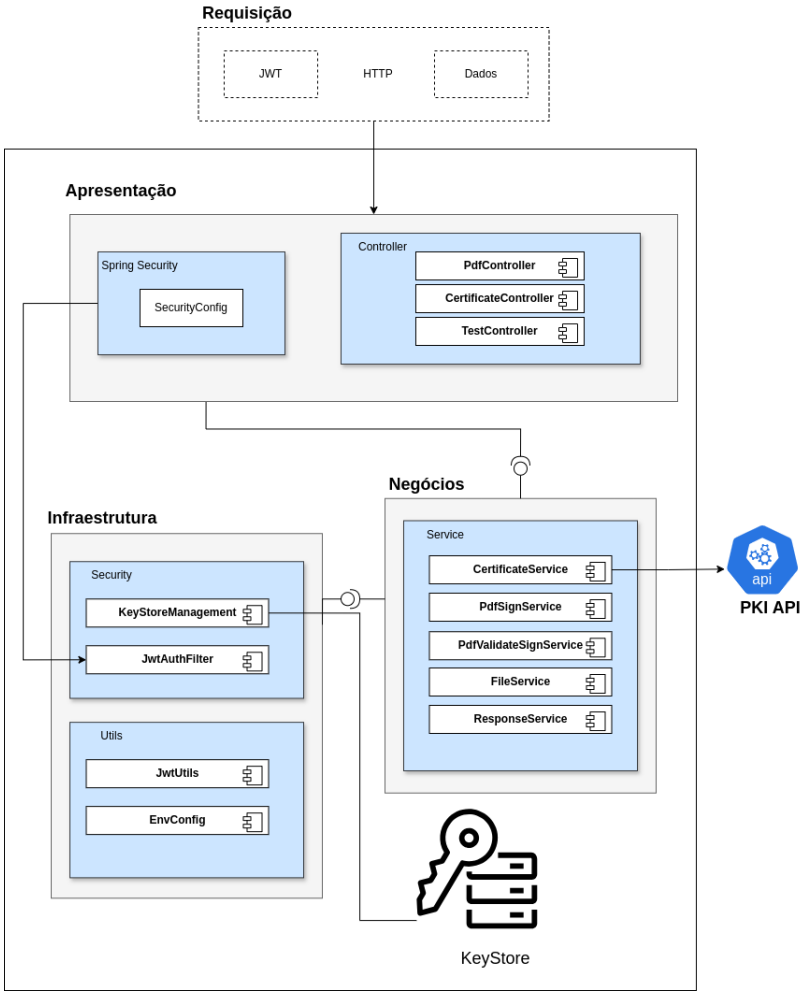
A arquitetura de software de um sistema consiste em diferentes estruturas para entender e analisar o sistema, o que inclui elementos de software e suas características, conforme é colocado por Bass e Clements (2012). Cada estrutura é composta por um conjunto de elementos e as interações entre eles. Uma visualização é uma maneira de representar um conjunto coerente desses elementos arquitetônicos, de modo que possa ser interpretada pelos envolvidos no sistema. Assim, uma visualização é a representação de uma ou mais dessas estruturas.

4.4.1 Arquitetura da API REST

A arquitetura em camadas da API foi projetada de forma que a estrutura está organizada de maneira a tornar intuitivo o entendimento do papel de cada componente, bem como a forma como eles interagem entre si.“Uma camada é uma ‘máquina virtual’ abstrata que fornece um conjunto coeso de serviços por meio de uma interface gerenciada. Camadas podem usar outras camadas de forma estritamente gerenciada.”(BASS; CLEMENTS, 2012). Esse modelo em camadas promove uma clara separação de responsabilidades, o que facilita a evolução do sistema, o diagnóstico de problemas e a implementação de novas funcionalidades. A Figura 15 apresenta o diagrama que ilustra essa arquitetura, destacando a disposição das camadas e seus respectivos componentes.

A API REST foi desenvolvida utilizando o framework Spring e o Spring Boot, com Java como linguagem principal e o servidor Tomcat para hospedar a aplicação. A arquitetura segue o padrão síncrono com Spring Web MVC, onde os controladores recebem as requisições e os serviços processam as lógicas de negócios, acessando o *keystore* para consultar os certificados digitais necessários para assinatura e validação. Toda a gestão de dependências é feita pelo Maven, proporcionando um ambiente organizado para a aplicação.

Figura 15 – Arquitetura em Camadas da API REST UnBSign



Fonte: Autor.

A camada de Apresentação é responsável por interagir diretamente com os usuários e expor os pontos de acesso da API. Ela é composta pelos *Controllers*, que recebem as requisições HTTP, tratam as entradas e retornam as respostas. Neste caso, temos três principais controllers:

- ***CertificateController***: gerencia operações relacionadas a certificados digitais. Ele oferece *endpoints* para gerar certificados autoassinados, excluir todos os certificados armazenados, buscar um certificado por ID, e emitir e assinar certificados.
- ***PdfController***: gerencia operações relacionadas à assinatura e validação de PDFs. Ele oferece dois *endpoints* principais: um para assinar documentos PDF, utilizando um arquivo enviado pelo usuário e inserindo uma assinatura digital em uma posição específica da página, e outro para validar a assinatura de um PDF.
- ***TestController***: oferece dois *endpoints* de teste: um GET para verificar se a API está funcionando corretamente, retornando uma mensagem de sucesso, e um POST

para receber um nome como parâmetro e responder com uma saudação personalizada, confirmando que a requisição foi bem-sucedida.

A camada de Negócios contém a lógica central da aplicação, realizando o processamento e a implementação das regras de negócio. Os *Services* são responsáveis por manipular os dados e interagir com as camadas inferiores. Os serviços mais importantes no contexto de assinatura digital são:

- ***CertificateService***: responsável pela criação, armazenamento e manipulação de certificados digitais. Ele gera pares de chaves RSA, cria solicitações de assinatura de certificado (CSR), envia à autoridade certificadora (PKI) para obter um certificado assinado e gerencia certificados assinados. Os certificados são armazenados em um *keystore* (um repositório seguro usado para armazenar chaves privadas e certificados digitais, permitindo a gestão e proteção desses ativos criptográficos);
- ***PdfSignService***: responsável por assinar digitalmente documentos PDF, utilizando um certificado digital armazenado em um *keystore*. Ele prepara a aparência da assinatura, posiciona o campo de assinatura no PDF, e aplica uma assinatura criptográfica utilizando a chave privada do usuário e o algoritmo SHA-256. O processo de assinatura é realizado com a biblioteca iText e a implementação de uma assinatura externa utilizando o BouncyCastle como provedor de criptografia;
- ***PdfValidateSignService***: responsável por validar assinaturas digitais em arquivos PDF. Ele carrega o arquivo PDF, verifica as assinaturas presentes e processa cada uma delas. Para validar a integridade, ele extrai e compara o valor do resumo (*hash*) da assinatura no CMS (*Cryptographic Message Syntax*) com o resumo calculado a partir do conteúdo assinado no PDF. Além disso, ele verifica o certificado associado à assinatura, extraindo informações como o CN (*Common Name*), número de série e data de assinatura. O serviço também valida o algoritmo de assinatura e fornece informações sobre a integridade da assinatura.

A camada de Infraestrutura trata dos aspectos técnicos e de suporte à aplicação. Ela não contém lógica de negócios, mas provê funcionalidades essenciais para o funcionamento da aplicação, como segurança e acesso a recursos externos. Os principais componentes dessa camada são:

- ***JwtAuthFilter***: filtro de autenticação responsável por verificar o token JWT em cada requisição. Ele garante que apenas usuários autenticados possam acessar os recursos da API;
- ***KeyStoreManager***: gerencia operações na *keystore*, permitindo carregar, armazenar, adicionar e remover certificados de maneira segura;

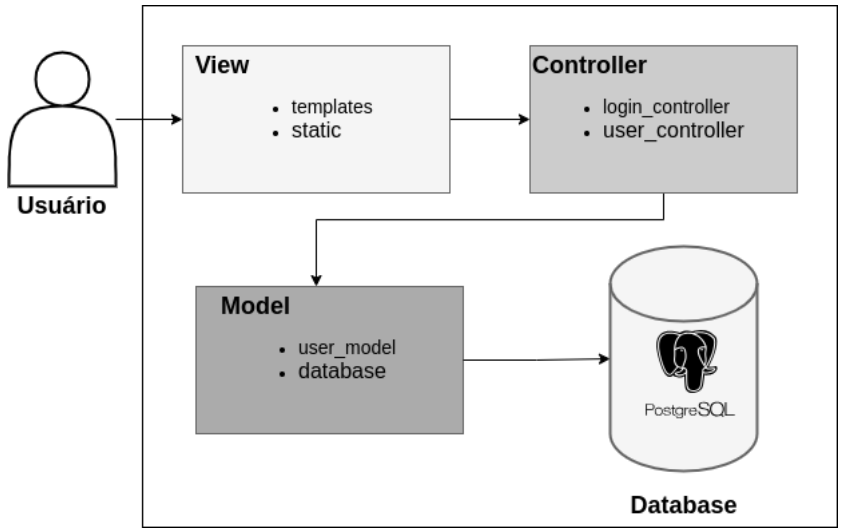
4.4.2 Arquitetura da *Web App*

A arquitetura da *Web App* segue o padrão MVC (*Model-View-Controller*), onde as interações do usuário são gerenciadas por uma estrutura modular em torno de uma aplicação *FastAPI*, com uma separação clara entre as responsabilidades de *front-end* e *backend*. O *backend* é responsável pela lógica de negócios, como a autenticação de usuários, gerenciamento de sessões e manipulação de arquivos. A aplicação utiliza rotas para mapear diferentes *endpoints* para renderizar páginas HTML ou processar dados via requisições POST. O diagrama da arquitetura pode ser visto na Figura 16.

Nesta separação de responsabilidades, as camadas tem as seguintes:

- ***Model***: responsável pela definição dos dados e da lógica de negócios. No caso da aplicação, os modelos representam os dados relacionados aos usuários e suas interações com o sistema
- ***View***: responsável pela apresentação dos dados ao usuário, e é composta pelos arquivos HTML que utilizam o mecanismo de templates Jinja2. Esses templates são usados para gerar dinamicamente as páginas que o usuário verá, como as telas de login, cadastro e upload de documentos. A visão é alimentada com dados do *Controller*, como o nome do usuário autenticado, e gera o HTML final para exibição. A separação entre lógica de apresentação (HTML, CSS, JavaScript) e lógica de controle permite que o front-end seja modificado sem impactar a lógica de negócios.
- ***Controller***: atua como intermediária entre a visão e o modelo. Ele recebe as requisições HTTP, executa a lógica de negócios necessária (como autenticação de usuários, criação de tokens, validação de credenciais) e interage com os modelos para obter ou manipular dados.

Figura 16 – Diagrama da Arquitetura MVC da *Web App*



Fonte: Autor.

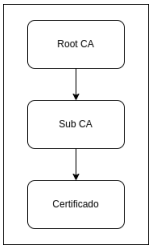
4.4.3 Arquitetura da API PKI

4.4.3.1 PKI Simples

A arquitetura da Infraestrutura de Chave Pública (PKI) foi desenvolvida com base em um tutorial simples de PKI disponível em openssl (2024).

O processo de construção da PKI começa com a criação de uma Autoridade Certificadora (CA) raiz simples e seu respectivo certificado de CA. A seguir, a CA raiz é utilizada para gerar uma CA de assinatura, que será responsável pela emissão de certificados para usuários finais. Esse modelo hierárquico permite uma gestão da segurança digital, onde a CA raiz estabelece a confiança, e a Sub CA é encarregada da emissão dos certificados de maneira controlada e delegada. Essa organização é apresentada na Figura 17.

Figura 17 – Hierarquia da PKI Simples



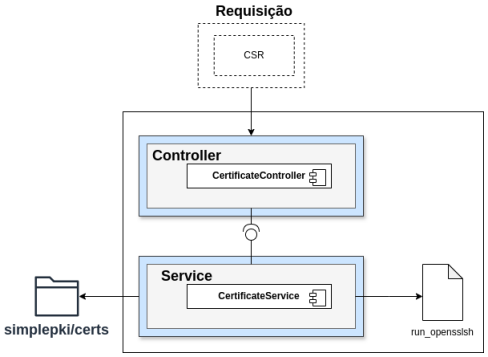
Fonte: Autor.

A estrutura de diretórios foi organizada para refletir essa hierarquia, com a CA raiz e a Sub CA, que se encarrega da emissão de certificados para as entidades finais (usuários ou sistemas). As entidades finais, por sua vez, são aquelas que recebem os certificados, permitindo-lhes autenticar sua identidade e garantir a segurança das comunicações.

4.4.3.2 Estrutura Spring Boot da PKI

A arquitetura da API PKI foi projetada utilizando uma abordagem semelhante à da API UnBSign, com a separação de responsabilidades entre os componentes Controller e Service. Sua diagramação pode ser vista na Figura 18.

Figura 18 – Diagrama de Arquitetura da API PKI



Fonte: Autor.

- ***CertificateController***: atua como o ponto de entrada para solicitações relacionadas à assinatura e validação de certificados. Ele expõe endpoints para processar requisições de assinatura de certificados via `/signature`, validação de certificados por meio de `/validate` e um endpoint de teste `/teste`. O controlador gerencia os erros e garante que respostas adequadas sejam enviadas ao cliente, retornando o certificado assinado ou os resultados da validação.

- ***CertificateService***: implementa a lógica de negócio associada à assinatura e validação de certificados. Para a assinatura, ele recebe o CSR e o *Common Name*, executa o processo utilizando `openssl` e retorna o certificado assinado. Além disso, oferece funcionalidade para verificar a validade de múltiplos certificados a partir de seus números de série, garantindo um retorno estruturado ao cliente.

5 Resultados

Este capítulo apresenta os resultados obtidos durante o desenvolvimento da API UnBSign, bem como das soluções complementares que a acompanham. Para ilustrar a interação entre os componentes do sistema, é apresentado um diagrama funcional, além da abordagem utilizada para o deploy, que foi realizada por meio da virtualização com Docker. Também são destacados os principais elementos da solução, incluindo a configuração de segurança, os serviços de assinatura e validação de documentos em formato PDF, e a estrutura da infraestrutura de chaves públicas (PKI).

Os repositórios estão no GitHub em uma organização chamada UnBSign que pode ser acessada pelo link: <https://github.com/orgs/UnBSign/repositories>.

Os links são:

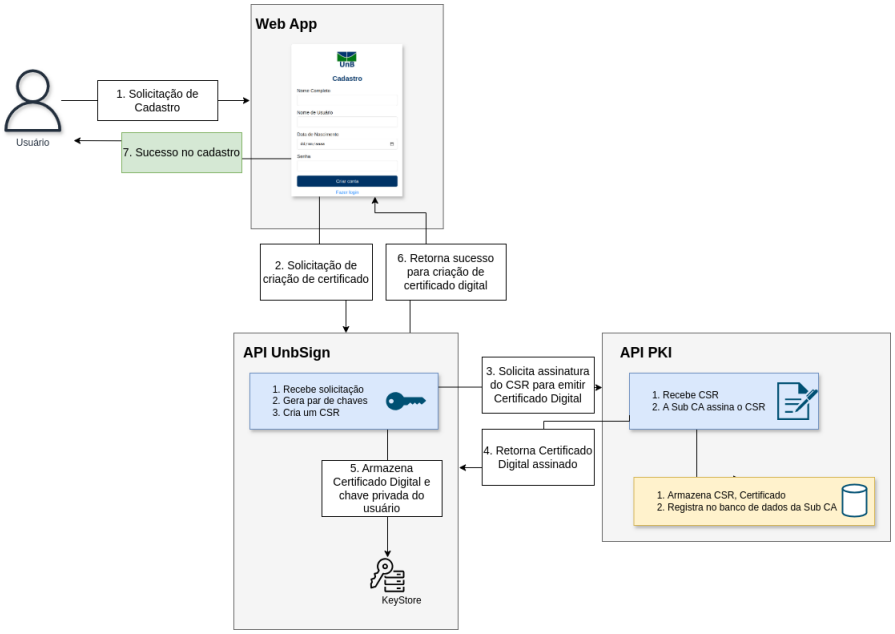
- UnBSign-API: <https://github.com/UnBSign/UnBSign-API>
- UnBSign-PKI: <https://github.com/UnBSign/UnbSign-PKI>
- UnBSign-WebApp: <https://github.com/UnBSign/UnBSign-WebApp>

5.1 Diagrama de Arquitetura Funcional

O diagrama de arquitetura funcional ilustra como os principais componentes interagem para realizar a funcionalidade de criação e utilização de certificados digitais. Ele demonstra o fluxo de informações entre o usuário, o *Web App* e as APIs (UnBSign e PKI). O foco está na integração entre as camadas do sistema, detalhando as etapas desde o cadastro do usuário e a geração do certificado digital até o uso desse certificado para assinar documentos PDF.

O primeiro diagrama, Figura 19, destaca o processo de cadastro do usuário e a geração do par de chaves, incluindo a criação de um CSR e a assinatura pelo PKI, além do armazenamento seguro das chaves e do certificado no *keystore*.

Figura 19 – Diagrama Funcional de Cadastro de Usuário



Fonte: Autor.

1. O usuário preenche os dados necessários na tela de cadastro do *Web App* e solicita a criação de uma conta;

2. *Web App* solicita à API UnBSign a criação de um certificado digital para o usuário;

3. A API UnBSign gera um par de chaves (pública e privada), cria um CSR, que contém as informações do certificado e envia o CSR para a API PKI assinar;

4. A API PKI recebe o CSR, assina o certificado com sua Sub-CA (subautoridade certificadora), armazena o CSR e o certificado assinado no banco de dados e retorna o certificado digital assinado para a API UnBSign.

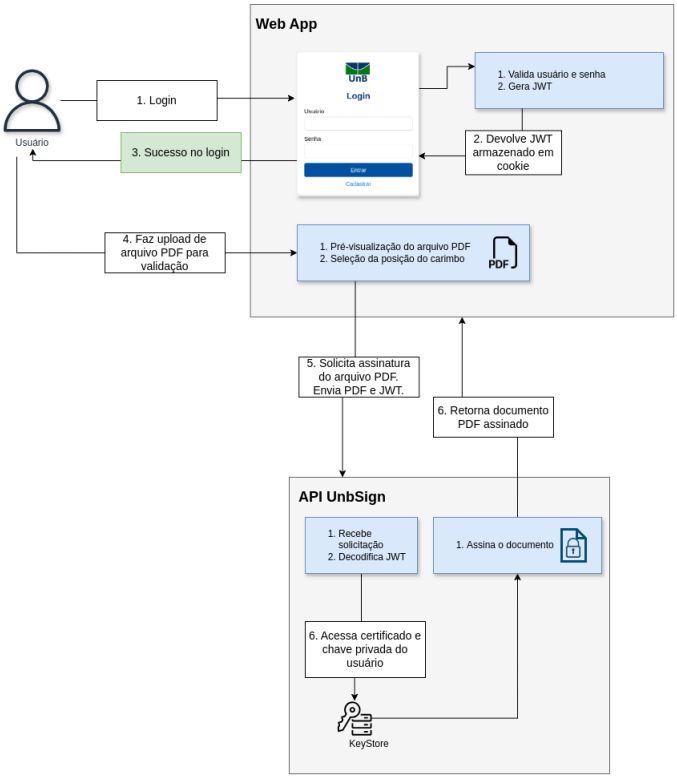
5. A API UnBSign armazena o certificado e a chave privada do usuário em uma keystore.

6. O *Web App* recebe a confirmação da API UnBSign sobre a criação do certificado e informa o sucesso ao usuário.

No segundo diagrama, Figura 20, é mostrado como o usuário, após login, envia um arquivo PDF para ser assinado digitalmente. O JWT gerado no login autentica o processo, garantindo a segurança. A API UnBSign acessa as credenciais do *keystore* para realizar a assinatura e retorna o documento assinado.

1. O usuário insere suas credenciais no *Web App*;
2. O sistema valida as credenciais e, se corretas, gera um JWT para autenticação. Após a autenticação, o JWT é armazenado em um cookie no navegador para futuras interações seguras;
3. Usuário autenticado pode enviar arquivo para assinatura;
4. O usuário carrega um arquivo PDF para assinatura. O sistema exibe uma pré-visualização do arquivo e permite a seleção da posição onde a assinatura será aplicada.
5. O *Web App* envia o arquivo PDF e o JWT para a API UnBSign, solicitando a assinatura.
6. A API UnBSign valida o JWT e decodifica os dados do usuário. Com acesso ao certificado digital e à chave privada armazenados no Keystore, a API realiza a assinatura do documento;
7. A API devolve o documento PDF assinado ao *Web App* e o usuário recebe o arquivo assinado como resposta;

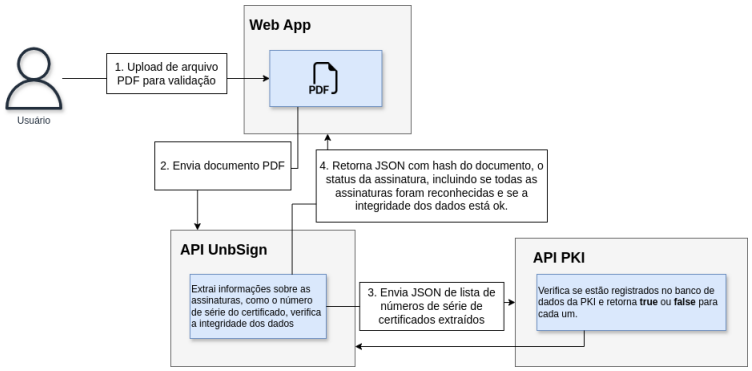
Figura 20 – Diagrama Funcional de Login e Assinatura de Documento



Fonte: Autor.

No terceiro diagrama, Figura 21, após o upload do documento, o *Web App* envia a solicitação à API UnBSign, que verifica a validade das assinaturas digitais e a integridade do conteúdo do documento. O resultado da validação é retornado ao usuário, confirmando se o documento permanece íntegro e devidamente assinado.

Figura 21 – Diagrama Funcional de Validação de Assinatura



Fonte: Autor.

1. O usuário realiza o upload de um arquivo PDF no *Web App* para verificar a validade das assinaturas digitais e a integridade do documento;

2. A aplicação web encaminha o PDF para a API UnBSign, responsável pela análise técnica das assinaturas digitais;

3. A API UnBSign extrai informações sobre as assinaturas digitais presentes no PDF, como o CN (Common Name) do certificado, número de série e data de assinatura. Verifica a integridade dos dados do documento para garantir que ele não foi alterado após ser assinado. Valida as assinaturas CMS e prepara um JSON com a lista de números de série dos certificados extraídos. Esse JSON é enviado para a API PKI.

4. A API PKI verifica se os números de série dos certificados extraídos estão registrados no banco de dados da PKI. Retorna para a API UnBSign um resultado indicando se cada certificado é válido ou não.

5. A API UnBSign consolida as informações recebidas da API PKI e retorna um JSON para o *Web App*. Esse JSON contém: O *hash* do documento, o status das assinaturas (válidas ou não), a confirmação da integridade dos dados do PDF.

6. Com base no JSON retornado pela API UnBSign, o *Web App* exibe para o usuário o status do documento, incluindo informações sobre a validade das assinaturas e a integridade do arquivo.

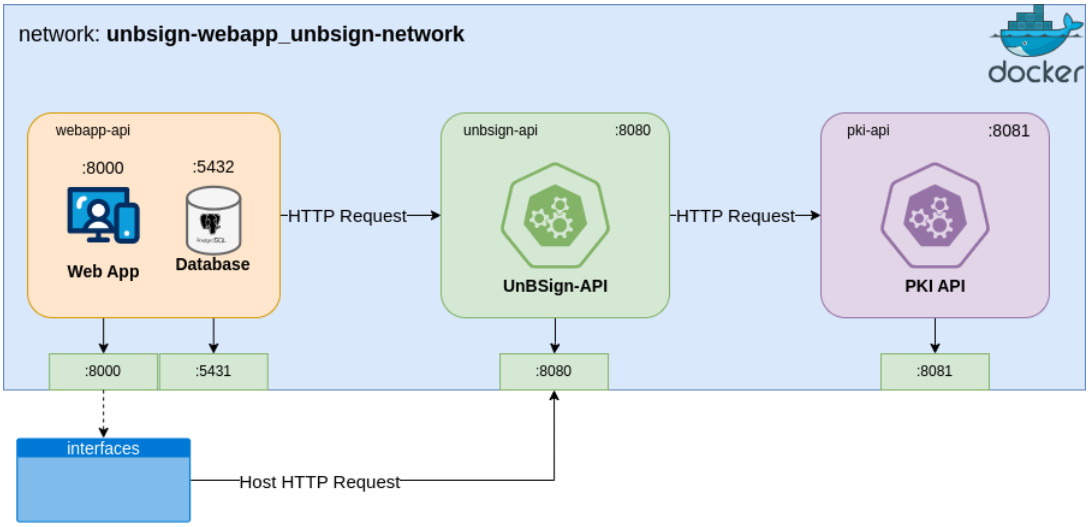
5.2 *Deploy* da Aplicação para Testes

A aplicação foi projetada para ser implantada utilizando containers Docker, proporcionando um ambiente virtualizado e eficiente para a realização de testes. Através do Docker Compose, a arquitetura do sistema é orquestrada de modo que cada serviço seja executado em containers independentes, o que assegura maior flexibilidade e escalabilidade. Embora os serviços estejam em containers distintos, todos estão conectados por uma rede comum, o que permite uma comunicação entre eles.

Todos os containers estão interligados pela rede *unbsign-webapp_unbsign-network*. Esta arquitetura facilita tanto o desenvolvimento quanto a realização de testes, ao mesmo tempo em que possibilita a escalabilidade e a integração de novos serviços.

A Figura 22 apresenta o diagrama de arquitetura do sistema, ilustrando como os diferentes serviços, executados em containers Docker, estão conectados por uma rede compartilhada.

Figura 22 – Arquitetura do Sistema Web com Containers Docker



Fonte: Autor.

- **WebApp (porta 8000)**: Servindo como a interface frontend da aplicação, o WebApp gerencia as interações do usuário e se comunica com o backend. Ele também se conecta ao banco de dados PostgreSQL na porta 5432;
- **Banco de Dados (PostgreSQL porta 5431)**: Armazena as informações utilizadas pelo WebApp.
- **UnBSign-API (porta 8080)**: A API principal do sistema, responsável por gerenciar as operações e integrar o WebApp com a PKI API. Ela recebe requisições do WebApp na porta 8080 e se comunica com a PKI API na porta 8081.
- **PKI API (porta 8081)**: Responsável pelas operações de infraestrutura de chave pública, como a assinatura digital e o gerenciamento de certificados. Ela é acessada pela UnBSign-API para realizar essas operações;
- **Interfaces Externas**: As interfaces externas são geradas pelo WebApp, que envia requisições para a UnBSign-API na porta 8080. Esse fluxo facilita a integração do sistema com outros serviços, permitindo que a aplicação se comunique com recursos externos de forma eficiente e segura.

A aplicação de teste pode ser acessada em <https://fcte.john.pro.br/unb-sign/login>. Caso o acesso não esteja disponível, pode haver interrupções temporárias devido a problemas técnicos.

5.3 API UnBSign

5.3.1 Configuração de Segurança

A configuração de segurança é definida pela classe *SecurityConfig*, onde a principal função é proteger os *endpoints* da aplicação utilizando o Spring Security. A principal característica é a configuração do filtro de autenticação JWT, que garante que as requisições sejam autenticadas via token de segurança.

O método *securityFilterChain* configura a proteção dos *endpoints*. O endpoint `/api/pdf/validation` é liberado para acesso público, enquanto `/api/pdf/signature` exige autenticação. A sessão é configurada para ser stateless, o que significa que o sistema não mantém sessões entre as requisições, dependente apenas do JWT para autenticação.

O filtro *JwtAuthFilter* valida o token JWT presente no cabeçalho *Authorization* de cada requisição. Se o token estiver ausente, for inválido ou expirado, a requisição é rejeitada com status *401 Unauthorized*. Caso contrário, o filtro extrai o identificador do usuário do token e o coloca no contexto de segurança, permitindo que a aplicação acesse os dados do usuário autenticado.

A classe *JwtUtils* é responsável pela manipulação do JWT, oferecendo métodos para extrair o identificador do usuário, validar o token, e decodificar o token para extrair suas reivindicações, utilizando a chave secreta para garantir a integridade do token.

Essa arquitetura de segurança garante que apenas usuários autenticados com um token válido possam acessar as funcionalidades protegidas da aplicação, como a assinatura de PDF e a validação de assinatura.

5.3.2 Assinatura e Validação de PDFs

5.3.2.1 Controlador *PdfController*

Este controlador tem como objetivo a implementação de funcionalidades para a assinatura e validação de arquivos PDF, utilizando a plataforma *Spring Boot*. Foram implementados dois *endpoints* principais que permitem, respectivamente, a assinatura digital de um arquivo PDF e a validação de sua assinatura digital.

- ***POST*** **`/api/pdf/signature`**: permite a assinatura de arquivos PDF. Ele recebe, na requisição, o arquivo PDF e informações sobre a posição e página onde a assinatura deve ser inserida. O identificador do usuário é extraído do contexto de segurança para vincular a assinatura ao usuário autenticado. O arquivo PDF é validado quanto ao formato de arquivo e integridade antes de ser assinado. A assinatura digital é aplicada com base nas informações de posição fornecidas, e, após o processo, o arquivo assinado é retornado ao usuário.

- ***POST*** **`/api/pdf/validation`**: foi desenvolvido para validar assinaturas digitais em arquivos PDF. O fluxo começa com a recepção de um arquivo PDF, que é então validado quanto à sua integridade. Em seguida, um serviço dedicado verifica a autenticidade da assinatura, confirmando se a assinatura é válida e se o arquivo não foi alterado após a assinatura. Por fim, o sistema retorna um conjunto de resultados detalhando a validade da assinatura.

5.3.2.2 Serviços *PdfSignService* e *PdfValidateSignService*

O serviço *PdfSignService* é responsável por assinar digitalmente documentos PDF utilizando um certificado digital. Ele oferece métodos para configurar a aparência da assinatura, adicionar um logo ao documento e determinar a posição da assinatura. O processo de assinatura envolve a extração do certificado e da chave privada de um *keystore*, a criação de uma assinatura digital usando o algoritmo de *hash* SHA256 e a inserção dessa assinatura no PDF.

O método `executeSign` é utilizado para orquestrar o processo de assinatura, carregando o *keystore*, validando o certificado do usuário e chamando o método *sign* para aplicar a assinatura digital no arquivo PDF. Suas funcionalidades são:

- `getCNFromX509Certificate(X509Certificate cert)`:
  Extrai o nome comum (CN) de um certificado X.509.
- `setAppearance(PdfSignatureAppearance appearance, String reason, String location, String certName)`: Define a aparência da assinatura digital, incluindo razão, local e um texto descritivo;
- `addLogoToAppearance(PdfSignatureAppearance appearance)`: Adiciona o logotipo à assinatura digital;
- `getSignatureRectangle(Float llx, Float lly, Float pageWidth, Float pageHeight)`: Calcula a posição e dimensões da assinatura dentro do PDF, garantindo que esteja dentro dos limites da página;
- `setDefaultSignatureRectangle()`: Define uma posição padrão para a assinatura caso não seja especificada;
- `setSignaturePosition(PdfReader reader)`: Analisa o PDF e armazena as coordenadas de texto para auxiliar na escolha da posição da assinatura;
- `getNextSignatureFieldName(PdfReader reader)`: Retorna o nome do próximo campo de assinatura no PDF, garantindo que seja único;

- `generateSignaturePolicy()`: gera informações sobre a política de assinatura digital, atribuindo um identificador único, calculando o hash do conteúdo da política utilizando um algoritmo de resumo criptográfico e associando um URL de referência.

- `sign(String src, String dest, Certificate[] chain, PrivateKey pk, String digestAlgorithm, String provider, CryptoStandard subFilter, String reason, String location, String certName, Integer pageNumber , Float posX, Float posY)`:
  Realiza a assinatura digital do PDF, posicionando a assinatura conforme as coordenadas informadas;

- `executeSign(String SRC, String DEST, String id, int pageNumber , Float posX, Float posY)`: Carrega o certificado do usuário do keystore e chama o método sign para assinar o PDF.

O fluxo de uso é:

1. O serviço recebe os parâmetros: caminho do PDF original, caminho do PDF assinado, identificador do certificado (id), número da página,e coordenadas do carimbo da assinatura;

2. O serviço carrega o *keystore*, o certificado correspondente ao id é recuperado e a chave privada associada ao certificado é extraída;

3. O nome comum (CN) do certificado é extraído para compor a assinatura e o serviço define os metadados da assinatura, incluindo a razão, local e nome do assinante. Se necessário, a posição da assinatura é ajustada automaticamente;

4. O serviço cria uma assinatura digital no documento. A aparência da assinatura é configurada (incluindo texto descritivo e logotipo da UnB). O PDF é assinado utilizando a chave privada e o algoritmo de *hash* SHA-256;

5. O serviço finaliza a assinatura e retorna o PDF assinado para uso;

Este serviço também utiliza as bibliotecas *iText* e *BouncyCastle*. Ambas as bibliotecas são amplamente utilizadas para manipulação de PDFs e para criptografia, sendo fundamentais para a implementação de assinaturas digitais.

As funcionalidades principais do *iText* permitem adicionar uma assinatura visual, sendo elas:

- `PdfReader`: Carrega o documento PDF para assinatura;

- `PdfStamper`: Aplica a assinatura ao documento carregado.

- `PdfSignatureAppearance`: Configura a aparência da assinatura, incluindo texto e imagem (logo);
- `MakeSignature`: Realiza a assinatura digital usando a chave privada e o certificado do usuário.

O método `MakeSignature.signDetached` do iTextPDF possibilita a criação de uma assinatura digital em um documento PDF de maneira desanexada. Isso significa que a assinatura é associada ao documento, sem modificar seu conteúdo original, sendo inserida em um bloco separado de assinatura. Tal abordagem garante que o conteúdo do PDF permaneça intacto, enquanto a assinatura digital valida a integridade e autenticidade do documento. O processo ocorre da seguinte forma:

1. **Geração do *hash* do documento**: O conteúdo do PDF é processado por meio do algoritmo de *hash* SHA-256, gerando um *hash* do documento;
2. **Assinatura do *hash***: O *hash* gerado no passo anterior é então assinado digitalmente utilizando a chave privada associada ao certificado do signatário;
3. **Configuração da aparência da assinatura**: A aparência visual da assinatura é configurada, permitindo a inserção de informações como o nome do signatário, data, localização e logo;
4. **Anexação da assinatura**: A assinatura digital resultante, junto com a cadeia de certificados (que inclui a chave pública para validação futura), é anexada ao documento no campo de assinatura designado;
5. **Inclusão da política de assinatura**: Essa política define regras e diretrizes que a assinatura deve seguir, incluindo o identificador único da política, o algoritmo de hash utilizado, e um URL que referencia a documentação da política.
6. **Geração do pdf**: O documento PDF resultante contém a assinatura digital no formato CMS (*Cryptographic Message Syntax*), preservando a integridade do conteúdo original do PDF.

O CMS (*Cryptographic Message Syntax*), descrito em R. Housley (2009), define um formato para proteger dados por meio de criptografia e assinaturas digitais. Ele permite que as informações sejam encapsuladas de diferentes maneiras, incluindo a possibilidade de aninhar camadas de proteção. Por exemplo, um dado pode ser assinado digitalmente e, em seguida, criptografado, ou vice-versa.

Além disso, ICP-Brasil (2021c) explica que o CMS possibilita que atributos adicionais sejam incorporados à assinatura, como o registro do momento em que a assinatura foi

realizada. Também permite a inclusão de contra-assinaturas e co-assinaturas, ampliando a flexibilidade do processo de autenticação. No entanto, ele não permite assinar apenas partes de um documento; a assinatura sempre se refere ao conteúdo como um todo.

Neste projeto, a implementação permite a adição de múltiplas assinaturas em um mesmo documento, viabilizando tanto co-assinaturas, onde diversos signatários assinam um documento de forma independente, quanto contra-assinaturas, onde uma assinatura é aplicada sobre outra previamente existente. Para isso, foi desenvolvido de forma a preservar assinaturas anteriores, garantindo que um novo processo de assinatura não invalide as assinaturas já presentes.

Já as funcionalidades utilizadas do *BouncyCastle* são:

- `BouncyCastleProvider`: Adiciona o provedor de criptografia à aplicação.
- `ExternalDigest`: Calcula o resumo da mensagem (*hash*) usando SHA-256.
- `ExternalSignature`: Define a assinatura digital com a chave privada do usuário e o algoritmo SHA-256.
- `PrivateKeySignature`: Implementa a assinatura digital usando a chave privada e o algoritmo SHA-256.
- `DigestAlgorithms`: Fornece algoritmos de *hash*, como o SHA-256, para garantir a integridade do documento.

Por sua vez, o serviço *PdfValidateSignService*, baseado em (SATRIO, 2022), é um serviço responsável pela validação de assinaturas digitais em documentos PDF. A principal função do serviço é verificar a integridade de um PDF assinado digitalmente, checando a validade da assinatura e o certificado associado. Suas principais funções são:

- `validateSignature(String filePath)`: valida as assinaturas digitais presentes no PDF, verificando a integridade do documento e a validade dos certificados associados. Retorna um mapa com informações sobre as assinaturas e o status de validação.
- `processSignature(PDSignature signature, ByteArrayInputStream pdfBytes)`:
  Processa uma assinatura individual, extraindo informações como o certificado, o algoritmo de digestão, a data de assinatura e verifica a integridade da assinatura.;
- `getCNFromCertificate(X509CertificateHolder certificateHolder)`: Extrai o Common Name (CN) do certificado digital, que geralmente contém informações sobre o proprietário do certificado.

- `getByteRangeData(ByteArrayInputStream bis, int[] byteRange)`:
  Extrai os dados do PDF que foram assinados, com base no intervalo de bytes especificado na assinatura.
- `validateCertificatesWithPKI(List<String> serialNumbers)`: Valida os certificados digitais com um serviço externo de PKI (Public Key Infrastructure), verificando se os números de série dos certificados são reconhecidos e válidos;
- `getDocumentHash(String filePath)`: Calcula o hash do documento PDF usando o algoritmo SHA-256.

E o fluxo de execução é o seguinte:

1. Recebe o arquivo PDF como parâmetro;
2. As assinaturas do documento são identificadas e processadas individualmente;
3. Verifica se a assinatura utiliza filtros suportados, como *Adobe.PPKLite* e *ETSI.CAdES.detached*;
4. O campo CN, número de série e chave pública são extraídos do certificado X.509;
5. O intervalo de bytes da assinatura é recuperado para validação;
6. O *hash* do conteúdo assinado é comparado com o *hash* armazenado para garantir integridade;
7. Validação da Assinatura: A assinatura é validada com algoritmos como SHA-1 ou SHA-256;
8. Os números seriais dos certificados são enviados à API PKI para validação;
9. O resultado da validação é retornado com informações sobre cada assinatura;

As principais bibliotecas utilizadas foram *Apache PDFBox, BouncyCastle.*

A biblioteca *Apache PDFBox* foi utilizada para manipulação e leitura de documentos PDF. Ela foi particularmente importante para extrair as assinaturas e possibilitar a análise detalhada de seus metadados, como filtro, subfiltro, informações de contato e motivo.

O *BouncyCastle* foi utilizado para lidar com estruturas e algoritmos criptográficos complexos necessários para a validação de assinaturas digitais. A biblioteca fornece suporte para o processamento de dados assinados no formato PKCS#7, verificação de informações do certificado digital (como o CN) e cálculo de atributos como o digest da mensagem. Com isso, foi possível validar a integridade dos dados e autenticar a origem da assinatura, além de extrair informações detalhadas do certificado.

5.3.3 Emissão e Controle de Certificados Digitais

5.3.3.1 Controlador *CertificateController*

O *CertificiateController* tem a finalidade de gerenciar operações relacionadas a certificados digitais, incluindo a criação, armazenamento, remoção e assinatura de certificados. Ele fornece *endpoints* para interagir com os serviços de certificados, utilizando o CertificateService para executar as operações de negócios. Abaixo, é explicado o funcionamento de cada um dos métodos do controlador:

- ***POST*** `/api/certificates/generate-self-signed`: cria um certificado digital autoassinado. Ao receber um objeto *CertificateRequest* com os parâmetros id e cn, ele chama o serviço *certificateService.createAndStoreCertificate* para gerar e armazenar o certificado no *keystore*. Em seguida, retorna uma mensagem confirmando a criação do certificado;
- ***DELETE*** `/api/certificates/delete-all`: este endpoint foi criado para ser usado em desenvolvimento. Ele exclui todos os certificados armazenados no *keystore*. Caso a operação seja bem-sucedida, retorna uma mensagem informando que todos os certificados foram deletados, ou uma mensagem de erro em caso de falha;
- ***GET*** `/api/certificates/certificate/id`: permite recuperar um certificado armazenado com base no identificador (id) fornecido na URL. Se o certificado for encontrado, é retornado com status 200; caso contrário, uma mensagem de erro é retornada com status 404.
- ***POST*** `/api/certificates/issue-certificate`: emite e assina um certificado. Primeiramente, ele gera um CSR, em seguida, o CSR é enviado a API PKI para assinatura. Se a assinatura for bem-sucedida, o certificado assinado é processado e armazenado. O método retorna uma mensagem de sucesso ou erro dependendo do resultado do processo de assinatura e armazenamento;

Os endpoints `/delete-all` e `/generate-self-signed` são recomendados para uso exclusivo em ambientes de desenvolvimento, pois permitem manipulações diretas e sem controle de certificados armazenados no *keystore*, o que pode representar riscos de segurança em ambientes de produção. O primeiro exclui todos os certificados, o que pode causar perda de dados críticos, enquanto o segundo cria certificados autoassinados, que não são validados por autoridades certificadoras confiáveis, comprometendo a confiança e a integridade do sistema em um ambiente de produção.

5.3.3.2 Serviço *CertificateService*

O serviço *CertificateService* é responsável por fornecer funcionalidades relacionadas à geração, armazenamento e processamento de certificados digitais. Ele usa a biblioteca BouncyCastle para manipulação de certificados X.509 e PKCS#10. Além disso, o serviço integra-se com a API PKI para assinatura de certificados. Aqui estão os métodos principais e como eles se conectam:

- **`createCsr(String id, String commonName)`**: Gera um CSR usando uma chave pública e privada RSA. Retorna o CSR codificado em Base64.
- **`generateCsr(PrivateKey privateKey, PublicKey publicKey, String commonName)`**: Cria o CSR usando a chave privada e pública fornecidas, com o nome comum especificado.
- **`createAndStoreCertificate(String id, String cn)`**: Gera um certificado autoassinado usando um par de chaves RSA e o nome comum especificado, e armazena o certificado no *keystore*.
- **`processAndStore(String id, String signedCertContent)`**: Processa um certificado assinado recebido em formato PEM, converte-o para X509 e o armazena no *keystore*;
- **`generateSelfSignedCertificate(KeyPair keyPair, String cn)`**: Gera um certificado X509 autoassinado a partir de um par de chaves RSA e um nome comum;
- **`deleteAllCertificates()`**: Remove todos os certificados do *keystore*;
- **`getCertificateByAlias(String alias)`**: Recupera um certificado do *keystore* pelo alias e o retorna como uma string no formato PEM;
- **`generateRsaKeyPair()`**: Gera um par de chaves RSA de 2048 bits;
- **`storeCertificate(String alias, X509Certificate certificate)`** : Armazena o certificado X509 no *keystore*, associado ao alias fornecido;
- **`convertPemToX509Certificate(String pem)`**: Converte um certificado no formato PEM para o formato X509;
- **`pkiSignCertificate(MultiValueMap<String, String> body, String id)`**: Envia uma solicitação para API PKI para assinatura do certificado. Após receber a resposta, o certificado assinado é processado e armazenado no *keystore*.

5.3.4 Referência de *Endpoints* da API

As Tabelas 5 e 6 apresentam os *endpoints* descritos anteriormente, suas respectivas funcionalidades e parâmetros necessários. Todos os *endpoints* seguem o padrão de URL iniciado com `/api/`.

Exceto pelo *endpoint* `pdf/validation`, todos os demais requerem o envio de um JWT no cabeçalho de autorização (*Authorization Header*) para autenticação e validação de acesso.

Tabela 5 – *Endpoints* da API UnBSign

Endpoint	Descrição
POST `/certificates/generate-selfsigned`	Gera um certificado digital autoassinado.
DELETE `/certificates/delete-all`	Exclui todos os certificados armazenados no *keystore*.
GET `/certificates/certificate/{id}`	Recupera o certificado armazenado pelo alias id.
POST `/certificates/issue-certificate`	Cria e assina um certificado, enviando o CSR a um serviço remoto.
POST `/pdf/signature`	Assina um arquivo PDF.
POST `/pdf/validation`	Valida a assinatura de um arquivo PDF.
GET `/test`	Endpoint de teste.
POST `/test`	Endpoint de teste com envio de um nome.

Fonte: Autor.

Tabela 6 – Parâmetros de Endpoint

Endpoint	Parâmetros/Corpo de Requisição
POST `/certificates/generate-selfsigned`	corpo contendo "id"e "cn"
DELETE `/certificates/delete-all`	-
GET `/certificates/certificate/{id}`	Parâmetro de caminho: id
POST `/certificates/issue-certificate`	corpo contendo "id"e "cn"
POST `/pdf/signature`	Corpo contendo o arquivo, posX, posY e pageNumber.
POST `/pdf/validation`	Corpo contendo o arquivo a ser validado.
GET `/test`	-
POST `/test`	Corpo: nome

Fonte: Autor.

5.3.5 Documentação da API com *Springdoc*

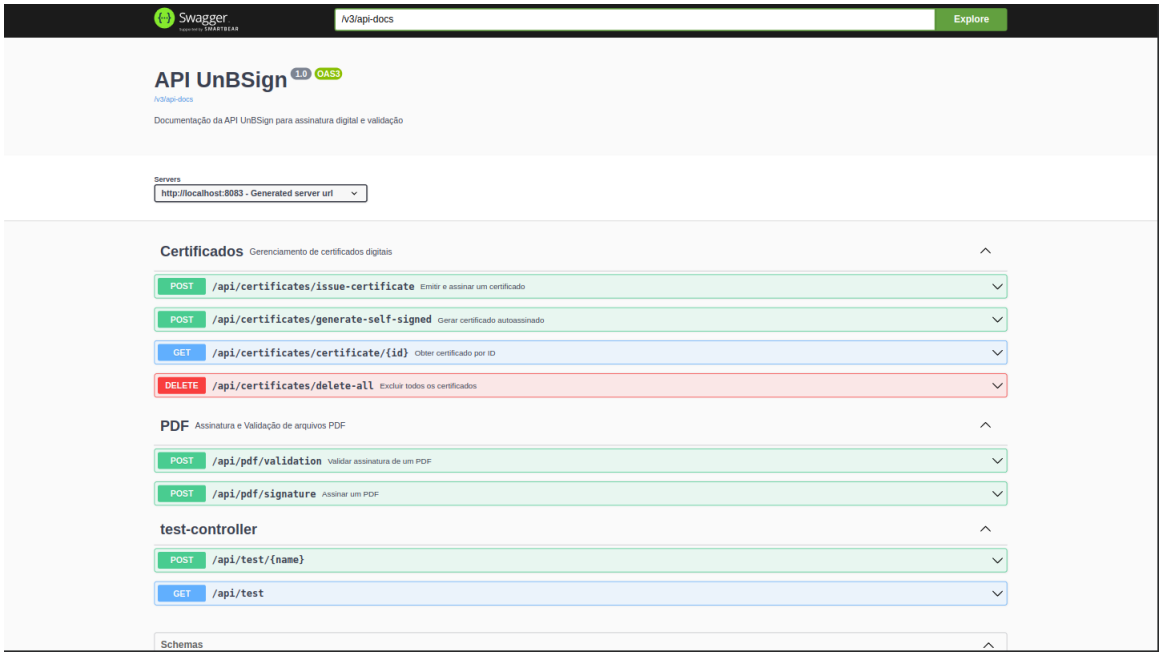
A documentação da API foi elaborada utilizando o *Springdoc*, uma biblioteca que simplifica a integração entre o Spring Framework e as especificações do Swagger e OpenAPI. O *Springdoc* possibilita a geração automática de documentação detalhada para APIs REST, extraindo informações diretamente do código-fonte da aplicação, eliminando a necessidade de criação manual de especificações. Essa abordagem oferece um método para documentar os endpoints da API, garantindo consistência e precisão.

Por meio do *Springdoc*, a documentação é construída com base nas anotações presentes nas classes e métodos da aplicação, fornecendo uma visão estruturada e detalhada

da organização dos serviços e das formas de interação com eles. Adicionalmente, o *Springdoc* integra-se de forma nativa ao Swagger UI, uma interface gráfica interativa que permite aos desenvolvedores explorar e testar os endpoints da API de maneira intuitiva e prática, facilitando a compreensão e a validação das funcionalidades disponíveis.

Após a aplicação ser iniciada com sucesso, a documentação da API estará disponível no navegador através da interface interativa do Swagger, permitindo visualizar e testar os endpoints da API. A interface pode ser acessada no *endpoint /swagger-ui.html*. Essa interface oferece uma visão dos recursos disponíveis e permite realizar chamadas diretamente para testar o funcionamento da API. A tela da interface interativa pode ser vista na Figura 23.

Figura 23 – Interface Interativa Swagger



Fonte: Autor.

5.4 API PKI

A construção da API PKI é dividida em duas etapas principais. A primeira etapa envolve a preparação do ambiente de certificação, abrangendo a criação e organização dos diretórios e arquivos essenciais para a infraestrutura de chave pública (PKI). Essa fase garante que todos os componentes necessários estejam devidamente estruturados para suportar a criação e gerenciamento dos certificados. A segunda etapa é dedicada ao desenvolvimento da API REST, que oferece um endpoint capaz de receber uma requisição de assinatura de certificado (CSR) e retornar o certificado assinado. A estruturação dos diretórios e arquivos da PKI, assim como a API REST para emissão de certificados, são detalhadas no Apêndice A.

5.5 *Web App*

A *Web App* foi desenvolvida com o objetivo de demonstrar o funcionamento da API UnBSign por meio de uma interface interativa. O backend foi construído utilizando FastAPI, responsável pela renderização de páginas HTML dinâmicas. A aplicação se conecta a um banco de dados PostgreSQL, que gerencia o cadastro de usuários, autenticação de login e a geração de tokens JWT. Esses tokens são utilizados para garantir a autorização e a comunicação segura com a API de assinatura, permitindo o acesso exclusivo aos serviços protegidos da API para usuários autenticados. No Apêndice B, encontra-se uma descrição detalhada sobre a modelagem e migração do banco de dados, além de informações sobre o backend, as telas de usuário e suas funcionalidades.

6 Conclusão

O objetivo principal deste estudo foi alcançado com a apresentação da API UnB-Sign, a qual possibilita a assinatura digital de arquivos PDF utilizando chaves privadas de certificados assinados por uma Infraestrutura de Chaves Públicas (PKI). Adicionalmente, foi desenvolvido um sistema PKI e uma aplicação Web que demonstram o funcionamento da solução, proporcionando uma visão prática e aplicada da proposta.

O desenvolvimento da API atendeu a todos os requisitos priorizados como *Must* na Tabela 4, além de implementar integralmente ou em parte, os requisitos *Should* e *Could*, conforme descrito a seguir:

- RFA14: A API suporta a inclusão de metadados adicionais na assinatura, como a hora exata da assinatura e o identificador do processo, garantindo a rastreabilidade;
- RFA15: A lógica de assinatura foi projetada de maneira modular, permitindo futuras atualizações ou suporte a diferentes algoritmos de assinatura sem afetar o funcionamento da API. Embora atualmente utilize apenas um algoritmo de assinatura, o design da API facilita a implementação de novos algoritmos no futuro;
- RFA18: A API responde em um formato JSON padronizado, facilitando a integração com sistemas externos e garantindo que mensagens de erro e sucesso sejam claras e compreensíveis;
- RFA25: A API retorna mensagens de erro claras caso ocorra algum problema com a assinatura ou a validação do documento (por exemplo, erro de formato, erro de assinatura ou token inválido);

Este trabalho, portanto, apresentou um MVP da aplicação, com foco na implementação inicial da assinatura digital. Dado o vasto contexto que envolve tanto aspectos tecnológicos quanto jurídicos, há um considerável potencial para futuras pesquisas e aprimoramentos. Esses avanços podem incluir o fortalecimento da segurança, garantindo que dados sensíveis sejam protegidos contra acessos não autorizados e ataques externos, e da robustez, assegurando que a API continue funcionando de maneira estável, mesmo em situações de alta demanda ou diante de falhas inesperadas, a ampliação da compatibilidade com diferentes tipos de certificados digitais, a integração com outros sistemas administrativos da universidade e a inclusão de novos tipos de documentos para assinatura. Tais melhorias proporcionarão uma solução cada vez melhor, adaptada às necessidades da instituição, e contribuirão para a promoção de maior segurança e praticidade nas transações digitais.

6.1 Dívidas Técnicas e Sugestões de Melhorias Futuras

Embora o desenvolvimento da API de assinatura digital tenha atendido aos objetivos iniciais, existem diversas áreas que podem ser aprimoradas para garantir maior eficiência, segurança e escalabilidade do sistema. As dívidas técnicas a seguir identificam pontos que precisam de ajustes e melhorias, enquanto as sugestões de melhorias futuras visam consolidar a solução como uma ferramenta robusta, segura e alinhada com as necessidades da Universidade de Brasília.

- **Melhoria na Arquitetura para Facilitar Manutenção e Integração de Novas Funcionalidades**: A arquitetura atual da aplicação pode ser aprimorada para facilitar a manutenção e a expansão. A utilização de padrões de design, como o *Strategy Pattern*, pode ser implementada para permitir a troca e adição de funcionalidades de forma mais flexível. Isso tornaria a aplicação mais modular, reduzindo o acoplamento e facilitando a integração de novas funcionalidades, como a adição de diferentes tipos de documentos ou suporte a novos algoritmos de assinatura;

- **Fortalecimento da Segurança da API**: A segurança da API é um aspecto crucial, especialmente no contexto de assinatura digital. Para melhorar a proteção dos dados em trânsito, é fundamental implementar TLS (*Transport Layer Security*), garantindo que as comunicações entre o cliente e o servidor sejam criptografadas e seguras. Além disso, deve-se considerar a implementação de mecanismos de autenticação robustos, como OAuth ou autenticação multifatorial, para aumentar ainda mais a segurança da aplicação.

- **Integração com Certificados Digitais da ICP-Brasil**: A compatibilidade com certificados digitais da ICP-Brasil deve ser uma prioridade para garantir que a API esteja alinhada com a infraestrutura de chaves públicas nacional. Isso exigirá ajustes na integração da API, de modo a permitir a validação e assinatura de documentos utilizando os certificados emitidos pela ICP-Brasil. Esta integração também ajudará a garantir que a solução esteja em conformidade com os padrões de segurança e regulatórios brasileiros.

- **Desenvolvimento e Consolidação de uma Política de Assinatura**: desenvolver uma política de assinatura robusta que atenda às necessidades de negócios da UnB. A política deve ser submetida à CA-Raiz para a obtenção de um Identificador Único (*Object Identifier* - OID), o que contribuirá para a integração com a infraestrutura de chaves públicas e permitirá a identificação precisa das assinaturas digitais realizadas dentro da UnB.

- **Melhorias e Desenvolvimento dos Testes da API**: é fundamental melhorar e desenvolver os testes da API para garantir que a solução mantenha a qualidade ao longo do tempo.

Referências

AWS. *O que é uma aplicação Web?* Amazon Web Service, 2020. Acesso em: 02 de dezembro de 2024. Disponível em: <https://aws.amazon.com/pt/what-is/web-application/>. Citado na página 33.

BASS, L.; CLEMENTS, P. *Software Architecture in Practice*. 3th. ed. USA: Addison-Wesley Professional, 2012. ISBN 9780321815736. Citado 2 vezes nas páginas 50 e 51.

BIEHL, M. *API Architecture*. 2th. ed. USA: API-University, 2015. Citado na página 32.

BOURQUE, P.; FAIRLEY, R. E. *SWEBOK V3.0*. [S.l.]: IEEE Computer Society, 2014. <https://ieeecs-media.computer.org/media/education/swebok/swebok-v3.pdf>. Citado 4 vezes nas páginas 35, 36, 40 e 42.

BRASIL. Medida provisória nº 2.200-2, de 24 de agosto de 2001. *Diário Oficial [da] República Federativa do Brasil*, Brasília, DF, 2001. Disponível em: <https://www.planalto.gov.br/ccivil_03/mpv/antigas_2001/2200-2.htm>. Citado na página 23.

BRASIL. Lei nº 14.063, de 23 de setembro de 2020. *Diário Oficial [da] República Federativa do Brasil*, Brasília, DF, 2020. Disponível em: <https://www.planalto.gov.br/ccivil_03/_Ato2019-2022/2020/Lei/L14063.htm#art5>. Citado 3 vezes nas páginas 14, 24 e 25.

Equipe TOTVS. *Assinatura digital é confiável? Como garantir a segurança para seus documentos*. 2024. Acessado em: 19 fev. 2025. Disponível em: <https://www.totvs.com/blog/gestao-para-assinatura-de-documentos/assinatura-digital-e-confiavel/>. Citado na página 14.

ETSI. *ETSI TS 102 778-1 V1.1.1: Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 1: PAdES Overview - a framework document for PAdES*. 2009. Acessado em: 20 jan. 2025. Disponível em: <https://www.etsi.org/deliver/etsi_ts/102700_102799/10277801/01.01.01_60/ts_10277801v010101p.pdf>. Citado 2 vezes nas páginas 29 e 30.

Git. *git*. [S.l.]: git, 2024. <https://git-scm.com/>. Citado na página 37.

GOV.BR. *Saiba mais sobre a Assinatura Eletrônica*. Ministério da Gestão e da Inovação em Serviços Públicos, 2020. Acesso em: 08 de outubro de 2024. Disponível em: <https://www.gov.br/governodigital/pt-br/identidade/assinatura-eletronica/saiba-mais-sobre-a-assinatura-eletronica>. Citado na página 14.

ICP-Brasil. Assinaturas digitais na icp-brasil. Intraestrutura de Chaves Públicas Brasileira, p. 29, 2015. Citado 3 vezes nas páginas 24, 25 e 26.

ICP-Brasil. Requisitos para geração e verificação de assinaturas digitais na icp-brasil; doc-icp-15.01. Intraestrutura de Chaves Públicas Brasileira, p. 18, 2015. Citado 2 vezes nas páginas 29 e 31.

ICP-Brasil. Política de assinatura icp-brasil padrão cades; doc-icp-15.04. Intraestrutura de Chaves Públicas Brasileira, p. 11, 2021. Citado na página 30.

ICP-Brasil. Requisitos mínimos para políticas de assinatura digital na icp-brasil. Intraestrutura de Chaves Públicas Brasileira, p. 17, 2021. Citado na página 30.

ICP-Brasil. Visão geral sobre assinaturas digitais na icp-brasil. Intraestrutura de Chaves Públicas Brasileira, p. 31, 2021. Citado 3 vezes nas páginas 28, 30 e 66.

ITI. *ICP-Brasil*. [S.l.]: Instituto Nacional de Tecnologia da Informação, 2017. <https://www.gov.br/iti/pt-br/assuntos/icp-brasil>. Citado na página 23.

ITI. *Certificado digital: o que é, para que serve e como fazer um documento virtual*. Instituto Nacional de Tecnologia da Informação, 2020. Acesso em: 18 de junho de 2024. Disponível em: <https://www.gov.br/iti/pt-br/assuntos/noticias/iti-na-midia/iti-na-midia-certificado-digital-o-que-e-para-que-serve-e-como-fazer-um-documento-virtual#:~:text=De%20acordo%20com%20o%20Instituto,cart%C3%A3o%20ou%20um%20token%20criptogr%C3%A1fico.> Citado na página 22.

KATZ, J. *Digital Signatures*. USA: Springer, 2010. ISBN 978-0-387-27711-0. Citado na página 13.

KUROSE, J. F.; ROSS, K. W. *Redes de computadores e a Internet*. 8th. ed. USA: Bookman, 2021. ISBN 8582605587. Citado 4 vezes nas páginas 13, 19, 20 e 21.

Lean Startup Glossary. *Lean Startup Glossary*. 2015. Acessado em: 19 fev. 2025. Disponível em: <https://www.totvs.com/blog/gestao-para-assinatura-de-documentos/assinatura-digital-e-confiavel/>. Citado na página 14.

MORESI, E. Metodologia de pesquisa. Universidade Católica de Brasília, p. 108, 2003. Citado na página 34.

NIST. *Glossary of Key Information Security Terms*. Estados Unidos, 2013. 223 p. Citado na página 19.

OPENSSL. *Simple PKI*. OpenSSL, 2024. Acesso em: 26 de dezembro de 2024. Disponível em: <https://pki-tutorial.readthedocs.io/en/latest/simple/>. Citado na página 55.

Origem da Palavra. *Palavra criptografia*. 2014. Disponível em: <https://origemdapalavra.com.br/palavras/criptografia/>. Acesso em: 31 de março 2024. Citado na página 18.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software*. [S.l.]: AMGH, 2021. Citado 2 vezes nas páginas 36 e 37.

R. Housley. Rfc 5662 - cryptographic message syntax (cms). RFC Editor, p. 56, 2009. Citado na página 66.

Rajesh Bondugula. *RSA sign and verify using Openssl : Behind the scene*. 2018. Acessado em: 19 fev. 2025. Disponível em: <https://medium.com/@bn121rajesh/rsa-sign-and-verify-using-openssl-behind-the-scene-bf3cac0aade2>. Citado 2 vezes nas páginas 26 e 27.

SATRIO, M. R. *PDF-Signature-Check*. 2022. Acessado em: 30 de dezembro de 2025. Disponível em: <https://github.com/rsatrio/PDF-Signature-Check/>. Citado na página 67.

SECTIGO. *Public and private keys in public key cryptography*. 2020. Disponível em: <https://www.sectigo.com/resource-library/public-key-vs-private-key>. Acesso em: 13 de abril de 2024. Citado na página 19.

Serasa. *Transformação digital e seus impactos na sociedade: como a tecnologia impulsiona comunidades?* 2024. Acessado em: 19 fev. 2025. Disponível em: <https://www.serasaexperian.com.br/conteudos/inovacao-e-tecnologia/transformacao-digital/>. Citado na página 13.

SIGUnB. *Apresentação - SIGUnB*. 2020. Disponível em: <https://portalsig.unb.br/sobre-o-sigunb/apresentacao>. Acesso em: 31 de março 2024. Citado na página 13.

SOUZA, I. P. M. de; NETO, B. B. Certificação digital: Conceitos e aplicações. Fatec Taquaritinga, p. 14, 2017. Citado na página 21.

STALLINGS, W. *Cryptography and Network Security*. 6th. ed. USA: Prentice Hall, 2013. ISBN 0-13-243310-9. Citado 5 vezes nas páginas 18, 19, 20, 21 e 26.

TANENBAUM, A.; FEAMSTER, N. *Redes de computadores*. 6th. ed. USA: Bookman, 2021. ISBN 8582605609. Citado 6 vezes nas páginas 17, 18, 21, 22, 23 e 26.

UNB. Instrucao da reitoria n. 0003/2016. *Portal SEI*, Brasília, DF, 2016. Disponível em: <https://portalsei.unb.br/wp-content/uploads/2025/01/Instrucao_da_Reitoria_n_003_2016.pdf>. Citado na página 14.

UnB. *Projeto UnBDigital*. 2024. Acessado em: 18 fev. 2025. Disponível em: <https://portalsei.unb.br/projeto-de-implantacao/>. Citado na página 13.

WIEGERS, K.; BEATTY, J. *Software Requirements*. 3th. ed. USA: Microsoft Press, 2013. ISBN 978-0-7356-7966-5. Citado na página 43.

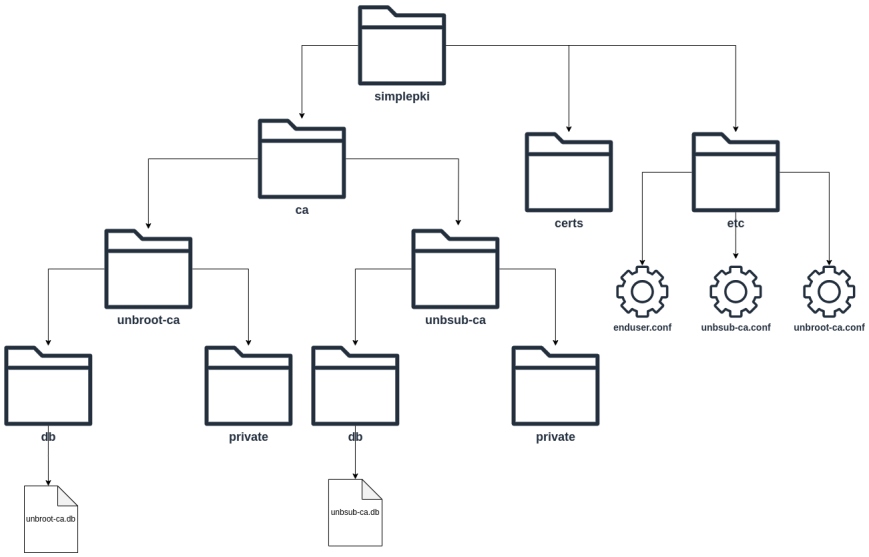
Apêndices

APÊNDICE A – Estruturação da PKI e API REST para Emissão de Certificados

A.1 Estruturação de diretórios e arquivos da PKI

Nesse processo, são criados os diretórios para armazenar chaves privadas, bancos de dados, CRLs (listas de revogação de certificados), e certificados emitidos, além de configurar as permissões adequadas de acesso. Também é gerado o banco de dados de controle de certificados e a numeração dos certificados e CRLs, garantindo que o sistema tenha um controle adequado e único de cada certificado emitido. A estrutura criada pode ser visto na Figura 24.

Figura 24 – Estrutura de diretórios da PKI



Fonte: Autor.

A estrutura de diretórios da PKI permite organizar e gerenciar os certificados e as chaves. O diretório *private* contém as chaves privadas da CA, *db* armazena o banco de dados de certificados e CRLs, *certs* guarda os certificados assinados e *etc* contém os arquivos de configuração da CA. Juntos, garantem a segurança e integridade do processo de emissão e validação dos certificados.

Os arquivos de configuração, localizados no diretório *etc*, são responsáveis por definir as regras e parâmetros para a criação e gestão de certificados digitais dentro da PKI. Cada arquivo de configuração é projetado para atender a uma função específica dentro da hierarquia da PKI, assegurando a segurança, integridade e rastreabilidade nas

operações de emissão, gerenciamento e validação de certificados.

O arquivo de configuração da unbroot-ca define os parâmetros para a autoridade certificadora (CA) raiz, como o tamanho da chave RSA, o uso de criptografia para proteger a chave privada, o algoritmo SHA-256 para assinatura, e as extensões do certificado. Também especifica os detalhes do Distinguished Name (DN) da CA, como nome da organização, país e nome comum. As configurações controlam a validade do certificado (3652 dias), a geração de números seriais, a política de nomeação, e as extensões de certificados, incluindo o uso de chaves para assinatura de certificados e listas de revogação (CRL).

O arquivo de configuração unbsub-ca é voltado para uma autoridade certificadora subordinada (CA de assinatura), responsável pela emissão de certificados para entidades finais. Ele define o uso de chaves RSA de 2048 bits, protege a chave privada como uso de criptografia e utiliza o algoritmo SHA-256. As extensões configuram o uso de chaves para assinatura de certificados e listas de revogação (CRL), e limitam a profundidade da cadeia de certificação, isto é, a unbsub-ca não pode emitir certificados para outras CAs (somente para entidades finais, como usuários ou servidores). Já o arquivo *enduser* configura a emissão de certificados para usuários finais, definindo parâmetros como o tamanho da chave, criptografia, algoritmo SHA-256, e as extensões necessárias para assinatura digital e criptografia de chave, com a adição de identificadores de chave para garantir a rastreabilidade.

Para a criação dos certificados e chaves necessárias, é utilizado o OpenSSL, sendo uma ferramenta robusta e amplamente usada para gerenciar certificados, criar requisições de certificados, gerar chaves privadas e públicas, e assinar certificados. O processo para construção da infraestrutura de certificação hierárquica e segura é o seguinte:

1. **Criação da Autoridade Certificadora Raíz (unbroot-ca)**: gerar um CSR, que inclui informações como o nome da autoridade, localização e propósito do certificado. Usando o OpenSSL com o comando *openssl req -new*, e o arquivo de configuração (*unbroot-ca.conf*), a requisição é gerada. O OpenSSL define parâmetros como o nome comum (CN) e o tamanho da chave. A requisição e a chave privada são salvas.

2. **Emissão do certificado autoassinado**: Após a criação da requisição de certificado, a unbroot-ca emite seu próprio certificado autoassinado, utilizando o OpenSSL com o comando *openssl ca -selfsign*. Isso gera o certificado *unbroot-ca.crt*, que é assinado pela própria chave privada da unbroot-ca, tornando-a uma autoridade confiável e raiz, capaz de assinar outros certificados na hierarquia de certificação.

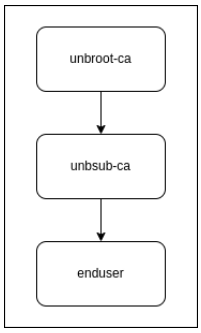
3. **Criação da Autoridade Certificadora Subordinada (unbsub-ca)**: gerar um CSR, que será usada para criar seu certificado. Isso é feito com o comando *openssl req -new*, utilizando o arquivo de configuração específico (*unbsub-ca.conf*), que define os parâmetros necessários, como o nome da autoridade e a chave pública.

4. **Emissão do certificado para a unbsub-ca**: é assinada pela unbroot-ca para estabelecer a hierarquia de confiança. O OpenSSL é utilizado, com o comando *openssl ca*, para assinar a requisição da unbsub-ca com a chave privada da unbroot-ca, gerando o certificado da unbsub-ca (unbsub-ca.crt), validando-a como uma autoridade subordinada confiável.

5. **Estabelecimento da hierarquia de confiança**: A assinatura do certificado da unbsub-ca pela unbroot-ca cria uma relação de confiança entre as duas autoridades certificadoras. Agora, a unbsub-ca pode emitir certificados para entidades finais (*endusers*) (como servidores, usuários ou sistemas), e esses certificados serão reconhecidos como válidos, pois foram assinados por uma autoridade confiável.

Esses passos estabelecem e possibilitam o gerenciamento de uma infraestrutura de certificação hierárquica e segura, conforme ilustrado na Figura 25. A utilização do OpenSSL facilita a geração de requisições de assinatura de certificados (CSR), a emissão de certificados digitais e a assinatura. Com base nos arquivos de configuração específicos para cada autoridade certificadora e para as entidades finais, o OpenSSL assegura a integridade, autenticidade e confidencialidade dos certificados emitidos, garantindo a segurança de toda a PKI.

Figura 25 – Hierarquia e respectivos componentes da PKI



Fonte: Autor.

Além disso, embora o OpenSSL suporte a revogação de certificados, essa funcionalidade não é empregada neste trabalho. A API PKI foi projetada com foco exclusivo na emissão de certificados a partir de CSRs (Certificado de Solicitação de Assinatura), sem a necessidade de gerenciar a revogação, como será detalhado nas próximas etapas.

A.2 API REST para Emissão de Certificados

A API PKI conta com um controlador, denominado *CertificateController*, que expõe o *endpoint api/pki/certificates/signature*. Além disso, a lógica de negócios é gerenciada pelo serviço *CertificateService*, responsável pelo processamento e assinatura dos certificados.

- ***POST*** `api/pki/certificates/signature`: expõe uma funcionalidade para assinar um certificado, recebendo duas informações principais: o CSR e o nome comum (*Common Name*). Ao receber esses dados, o serviço responsável processa a requisição, assina o certificado e retorna o certificado assinado como resposta. Em caso de erro durante o processo, uma mensagem de erro é retornada com um status apropriado.

- ***POST*** `api/pki/certificates/validate`: valida uma lista de certificados com base em seus números de série. Ele recebe uma lista de números de série no corpo da requisição e retorna um mapa com os resultados (número de série e status de validação).

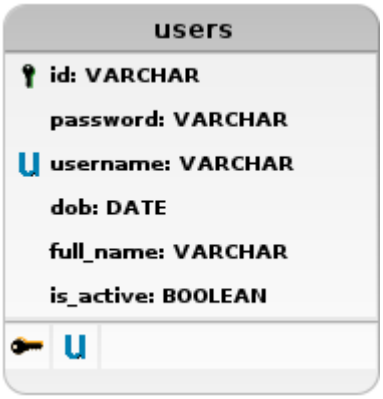
- ***CertificateService***: é responsável por assinar certificados e validar números de série. Para assinar o CSR e gerar o certificado assinado, recebe o CSR e o nome comum, cria o arquivo correspondente no diretório da PKI e executa a assinatura com o OpenSSL. Esse processo é automatizado por um script bash, que fornece a senha e confirma as etapas necessárias. Após a assinatura, o certificado gerado é extraído e retornado ao usuário. Caso haja qualquer erro durante o processo, uma exceção é gerada. Para validação, o serviço verifica se os números de série fornecidos existem no arquivo de banco de dados de certificados da unbsub-ca, retornando um mapa com o status de cada certificado. Se a lista de números de série for inválida, uma exceção é lançada.

APÊNDICE B – Descrição Modelagem de Banco de Dados e do Backend para *web app*

B.1 Modelagem e Migração de Banco de Dados

A tabela utilizado foi de usuários ("*users*"), que pode ser visto na Figura 26 abaixo.

Figura 26 – Tabela *users*



Fonte: Autor.

O dicionário de dados na Tabela 7 descreve a estrutura da tabela *users* utilizada no projeto. Ele fornece informações detalhadas sobre cada campo, seu tipo de dado, e as restrições impostas, como a obrigatoriedade de certos campos e a unicidade de outros.

Tabela 7 – Dicionário de Dados

Atributo	**Tipo de Dado**	**Descrição**	**Requisitos**
id	VARCHAR	Identificador UUID	Chave primária, Não Nulo
full_name	VARCHAR	Nome completo do usuário	Não Nulo
username	VARCHAR	Nome de usuário único para login	Único, Não Nulo
dob	DATE	Data de nascimento do usuário	Não Nulo
password	VARCHAR	Senha do usuário, armazenada em formato *hash*	Não Nulo
is_active	BOOLEAN	Indica se o token do usuário está ativo	Não Nulo

Fonte: Autor.

O modelo de dados da tabela foi definido utilizando o *SQLAlchemy*, uma biblioteca que oferece uma abstração para a criação e manipulação de bancos de dados relacionais de forma segura e eficiente. O uso do *SQLAlchemy* ajuda a prevenir vulnerabilidades de segurança, como o SQL injection. Isso ocorre porque a biblioteca utiliza *prepared statements* e *vinculação de parâmetros*, o que evita a inserção de código malicioso nas consultas

SQL. Em vez de construir consultas SQL dinâmicas com concatenação de strings, o *SQLAlchemy* cria as instruções SQL de forma segura, substituindo os parâmetros por valores devidamente escapados, minimizando o risco de ataques.

Além disso, as migrações do banco de dados são gerenciadas pelo *Alembic*, uma ferramenta que permite realizar alterações no esquema do banco de dados de maneira controlada e versionada. O *Alembic* facilita a evolução do banco de dados, permitindo que alterações no modelo de dados sejam refletidas nas tabelas reais.

B.2 *Backend*

O *backend* é responsável por gerenciar a lógica de autenticação de usuários, interagir com o banco de dados e realizar a comunicação com a API de assinatura digital.

O sistema utiliza para renderizar páginas HTML dinâmicas, com base na interação do usuário. A aplicação monta o diretório static para arquivos estáticos, como CSS e JavaScript, e o diretório templates para armazenar os arquivos de templates Jinja2. As páginas HTML são renderizadas com base nas rotas definidas:

- `/signature/upload`: Exibe a página de upload de documentos PDF para assinatura. A página é renderizada apenas se o usuário estiver autenticado, o que é validado por meio de um token de autenticação.
- `/validation/upload`: Exibe a página para validação de documentos assinados.
- `/cadastro`: Página de cadastro de novos usuários.
- `/login`: Página de login, onde os usuários inserem suas credenciais.

A autenticação no sistema é gerenciada utilizando tokens JWT (JSON *Web Token*). Quando o usuário faz login com suas credenciais (nome de usuário e senha), o sistema gera um token de autenticação utilizando uma chave secreta. Esse token contém informações essenciais, como o emissor (webapp_test), o ID do usuário, o nome de usuário e o tempo de expiração (definido para 1 hora).

Este token JWT é utilizado para validar o usuário em sessões subsequentes, garantindo que apenas usuários autenticados possam acessar funcionalidades restritas, como o upload de documentos para assinatura.

A utilização do JWT oferece uma autenticação sem estado, o que torna o sistema mais escalável e seguro. Isso ocorre porque o token contém todas as informações necessárias para verificar a identidade do usuário, eliminando a necessidade de consultar o banco de dados a cada requisição.

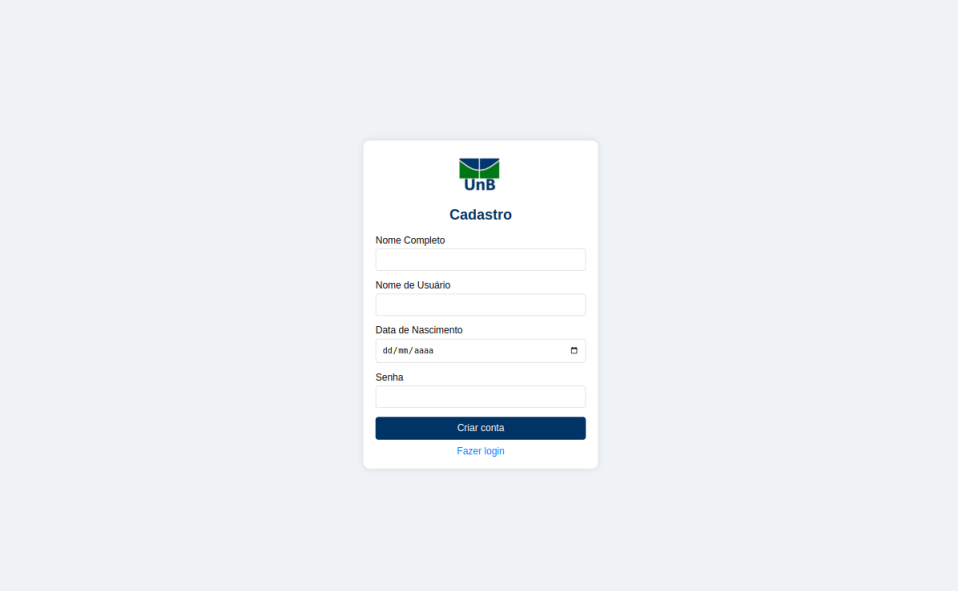
Ao realizar o logout, o token é invalidado e removido dos cookies, e o atributo *is_active* do usuário no banco de dados é atualizado para *false*, desautenticando o usuário e encerrando sua sessão de forma eficaz.

B.3 Telas do Usuário

A *Web App* possui quatro telas de usuários, renderizadas a partir de requisições *GET* ao *backend*, sendo que uma delas exige autorização para acesso. As telas incluem Cadastro, Login, Upload de Arquivo para Assinatura e Upload de Arquivo para Validação. As requisições do cliente ao *backend* ou à API de assinatura são realizadas com a Fetch API, uma interface JavaScript que permite fazer requisições HTTP assíncronas de forma simples e eficiente.

A tela de cadastro, como pode ser vista na Figura 27, apresenta um formulário simples que solicita os seguintes dados do usuário: nome completo, nome de usuário, data de nascimento e senha. Esses campos são necessários para o cadastro do usuário no sistema, permitindo que ele tenha acesso aos recursos e funcionalidades da aplicação.

Figura 27 – Tela de cadastro de usuários



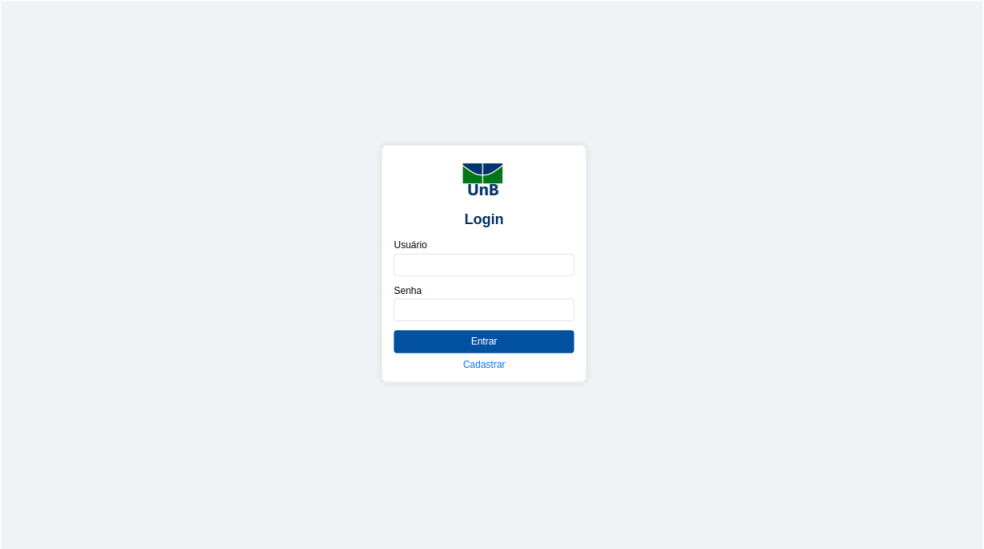
Fonte: Autor.

O código JavaScript nesta tela é responsável por capturar a submissão do formulário e realizar a requisição para o *backend* de forma assíncrona, evitando a recarga da página. Ao submeter o formulário, ele coleta os dados inseridos no formulário e os converte para um objeto JSON. Em seguida, uma requisição POST é feita para a rota `/cadastro` com o corpo da requisição contendo os dados do usuário. Caso a resposta seja bem-sucedida (status HTTP 201), o usuário é redirecionado para a tela de login. Se ocorrer um erro (status HTTP 422 ou outro), uma mensagem de erro é exibida ao usuário,

informando sobre o problema no processo de cadastro. Caso haja falha na requisição, um alerta é exibido, informando que ocorreu um erro ao tentar cadastrar o usuário.

A tela de login, como mostrado na Figura 28, solicita que o usuário forneça seu nome de usuário e senha previamente cadastrados no sistema. Esses dados são essenciais para validar a identidade do usuário e permitir o acesso às funcionalidades restritas (no caso, a página de assinatura).

Figura 28 – Telas de Login

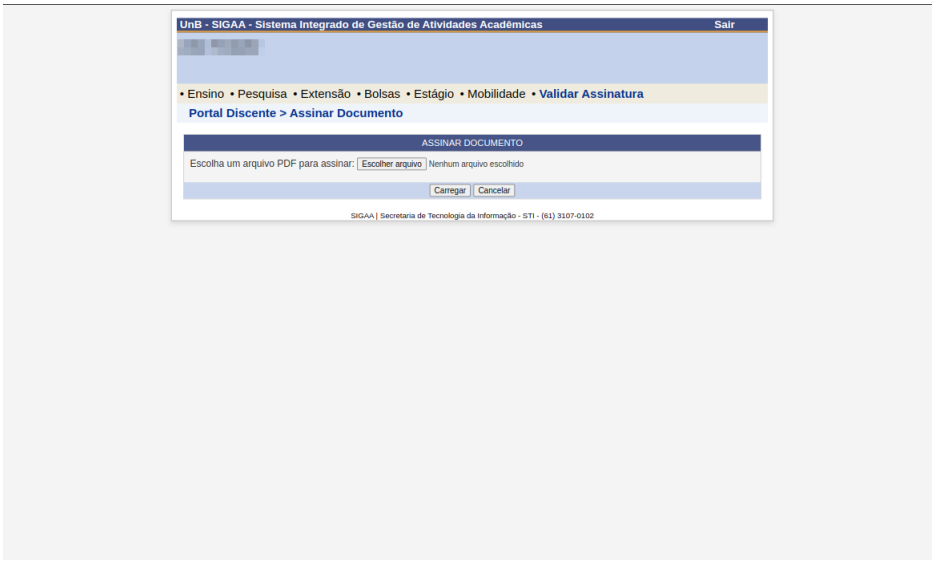


Fonte: Autor.

O código JavaScript associada ao formulário de login captura a submissão do formulário e evita o comportamento padrão de recarregar a página. Ele extrai os valores do nome de usuário e da senha inseridos no formulário, e envia esses dados para o *backend* via uma requisição POST. Caso a autenticação seja bem-sucedida (resposta HTTP 200), o sistema tenta acessar a página de upload de documentos para assinatura. Se essa solicitação também for bem-sucedida, o usuário é redirecionado para a página de upload. Caso contrário, é exibida uma mensagem de erro no console. Se as credenciais fornecidas forem inválidas, o usuário será alertado e solicitado a tentar novamente. O código também lida com erros inesperados, exibindo mensagens adequadas no console caso ocorra algum problema durante o processo de login.

A tela de upload de arquivos para assinatura, ilustrada na Figura 29, é acessível exclusivamente para usuários autenticados e autorizados. Para visualizar essa tela, o cliente deve apresentar um cookie denominado *authToken*, fornecido pelo *backend* no momento da autenticação. Além disso, a tela inclui um botão "**Validar Assinatura**", que redireciona para o endpoint responsável por verificar a autenticidade de assinaturas em documentos.

Figura 29 – Tela de upload de arquivos para assinatura



Fonte: Autor.

O código JavaScript gerencia o envio e a assinatura de arquivos PDF. O script verifica se o usuário enviou um arquivo válido e, em caso positivo, exibe o documento para visualização e manipulação, como visto na Figura 30. A assinatura é realizada com base na posição de um carimbo digital, que pode ser ajustado dinamicamente pelo usuário dentro do PDF. Além disso, a tela permite a navegação entre as páginas do documento e armazena temporariamente os arquivos no sessionStorage para facilitar a manipulação antes do envio.

Figura 30 – Tela de *upload* para assinatura com apresentação do arquivo e posicionamento de carimbo

Fonte: Autor.

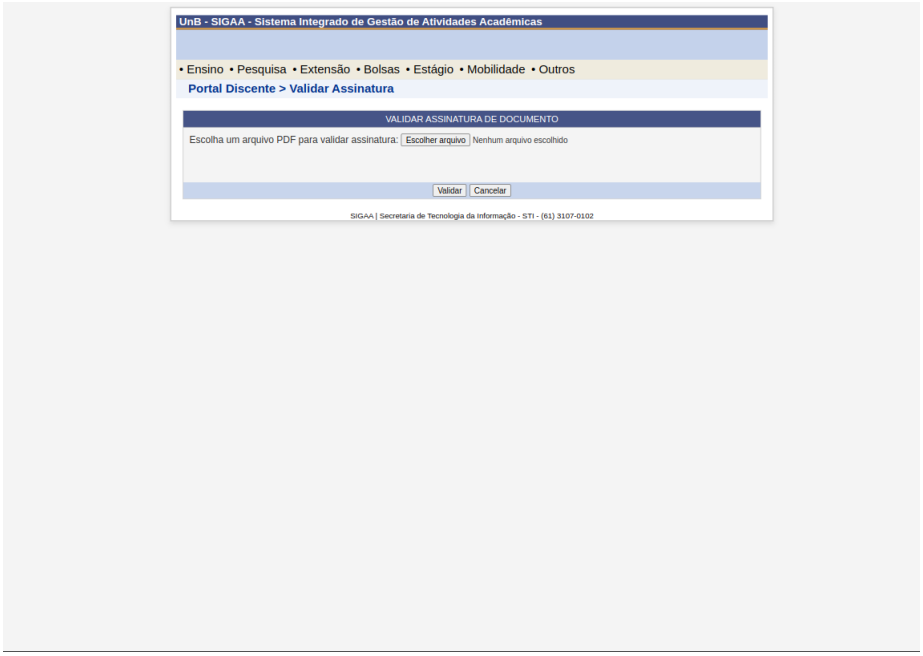
Ao clicar no botão "Assinar", o sistema verifica se o arquivo PDF foi carregado corretamente e se a posição do carimbo foi determinada. Em seguida, o arquivo é enviado para a API UnBSign para assinatura, incluindo o token de autenticação (*authToken*) no cabeçalho da requisição para garantir que apenas usuários autenticados possam assinar. Caso a assinatura seja bem-sucedida, o arquivo assinado é gerado e pode ser baixado pelo usuário. Em caso de erro, uma mensagem de falha é exibida. Um exemplo de carimbo de assinatura pode ser visto na Figura 31.

Figura 31 – Carimbo de Assinatura em um Documento

Fonte: Autor.

A tela de upload de arquivos para validação, mostrado na Figura 32 está disponível sem a necessidade de autenticação e permite que o usuário faça o upload de um arquivo PDF para ser validado. O objetivo dessa funcionalidade é apresentar o resultado da validação, identificando o titular do certificado digital utilizado em cada assinatura presente no documento. Caso o PDF contenha múltiplas assinaturas, o sistema fornecerá as informações relativas a cada uma delas. Além disso, o sistema verifica a integridade do documento, confirmando se ele não sofreu qualquer adulteração após a assinatura, garantindo que o conteúdo do PDF permaneceu intacto desde o momento em que foi assinado digitalmente.

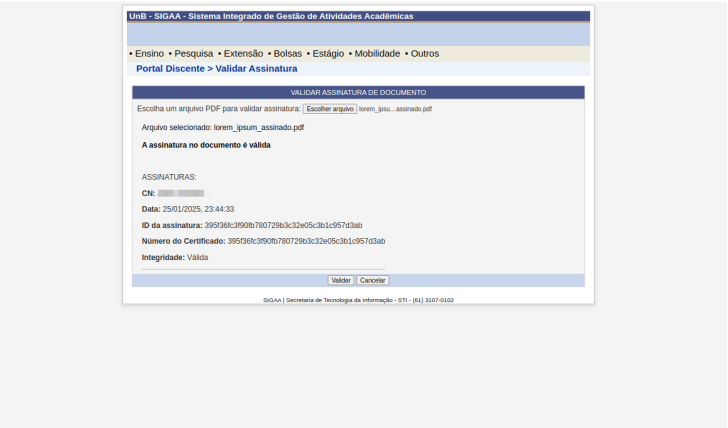
Figura 32 – Tela de *upload* de arquivos para validação



Fonte: Autor.

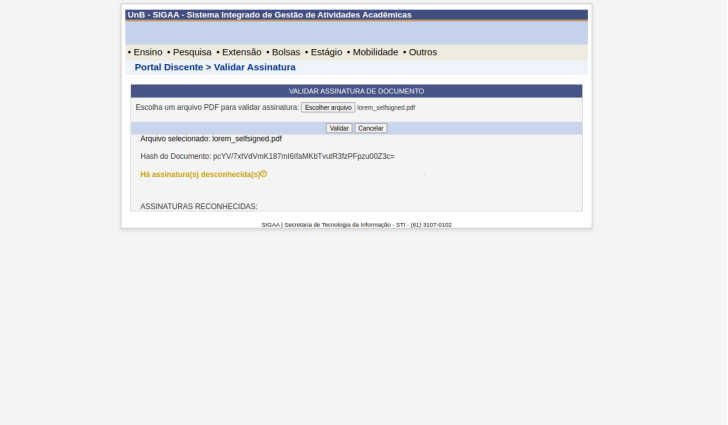
O código JavaScript gerencia o envio e a apresentação dos resultados da validação de arquivos PDF. Após a seleção do arquivo, o sistema envia o arquivo para o servidor por meio de uma requisição assíncrona, utilizando a API Fetch. O servidor então valida o arquivo e retorna um resultado indicando se a assinatura no PDF é válida. Caso a assinatura seja válida, detalhes sobre as assinaturas presentes no documento são exibidos, incluindo informações como nome do assinante, data, ID da assinatura, número do certificado e o estado de integridade da assinatura. Se houver assinaturas desconhecidas, uma indicação visual é fornecida, e a mensagem é ajustada para alertar o usuário sobre a assinatura não reconhecida pela PKI. Caso nenhuma assinatura seja encontrada, uma mensagem informando isso é exibida. Se a assinatura não for válida, uma mensagem de erro é apresentada, destacando que a assinatura é inválida. Em qualquer erro de comunicação ou processamento com o servidor, o sistema exibe uma mensagem com detalhes sobre o erro. Os diferentes resultados podem ser visualizados nas Figuras 33a, 33b, 33c e 33d.

Figura 33 – Tela de *upload* de resultados para validação de arquivos



(a) Resultado de validação do arquivo para assinatura válida

(b) Resultado de validação do arquivo para assinatura inválida



(c) Resultado de validação do arquivo com assinaturas não reconhecidas

(d) Resultado de validação do arquivo sem assinaturas
Fonte: Autor.

Anexos

ANEXO A – Política de Assinatura

A.1 Políticas de Assinatura [1]

A.1.1 Identificador da Política de Assinatura

A.1.1.1 Nome da Política de Assinatura

O nome desta Política de Assinatura é POLÍTICA DE ASSINATURA DIGITAL PARA VALIDAÇÃO DE DOCUMENTOS ELETRÔNICOS, versão 1.0

A.1.1.2 *Object Identifier (OID)*

O Object Identifier (OID) desta PA, atribuído pelo Instituto Nacional de Tecnologia da Informação (ITI) é: 2.16.76.1.5.1.1.1.

A.1.1.3 Novas Versões

Novas versões desta política, se criadas, receberão OID diferenciado, do tipo 2.16.76.1.5.1.1.n+1.

A.1.1.4 Proteção contra Alterações Indevidas

Esta PA está protegida contra alterações indevidas por meio da publicação, no repositório da AC Raiz da PKI UnB (<http://www.unb.gov.br>) do seu conteúdo assinado digitalmente por chave privada associada a certificado digital do Instituto Nacional de Tecnologia da Informação, utilizando algoritmo SHA256 com RSA.

A.1.2 Data da Criação

A data de criação desta PA é 25.01.2025.

A.1.3 Entidade Criadora da Política de Assinatura

A entidade criadora desta PA é a Universidade de Brasília (UnB).

[1] Os tópicos *Object Identifier (OID)*, Novas Versões, Proteção contra Alterações Indevidas, Entidade Criadora da Política de Assinatura, Período para Assinatura, Raiz Confiável, Conjunto de Políticas de Certificado Aceitável e Forma de Verificação do Status da Cadeia de Certificação (Revogação) não se aplicam ao contexto real, sendo utilizados somente de forma didática para este trabalho.

A.1.4 Campo de Aplicação

A.1.4.1 Aplicabilidade

Esta PA se aplica às assinaturas digitais de transações eletrônicas ou outras situações em que as informações de verificação das assinaturas digitais, necessárias para dirimir eventuais contendas futuras, podem ser obtidas por outras formas, como, por exemplo, podem ser providas por uma das partes, que as tenha registrado em seu sistema. Nessas situações, é recomendável que exista um acordo prévio, assinado por ambas as partes, concordando com essa guarda unilateral de dados complementares.

A.1.4.2 Utilização Geral

Ela pode ser utilizada por qualquer pessoa física ou jurídica, órgão de governo ou outro tipo de entidade, sem restrições.

A.1.4.3 Assinaturas Múltiplas

Segundo esta PA, é permitido o emprego de múltiplas assinaturas.

A.1.5 Política de Validação da Assinatura

A.1.5.1 Período para Assinatura

Esta PA terá validade de 25.01.2025 até 25.01.2027.

A.1.5.2 Regras Comuns

A.1.5.3 Regras de Signatário e Verificador

A.1.5.4 Regras do Signatário

A.1.5.4.1 Dados Externos ou Internos à Assinatura

As assinaturas realizadas segundo esta PA podem ser do tipo *attached* (que inclui o conteúdo assinado na assinatura digital) ou *detached* (que não inclui o conteúdo assinado na assinatura digital).

A.1.5.4.2 Atributos Assinados Obrigatórios

As assinaturas feitas segundo esta PA definem como obrigatórios os seguintes atributos assinados:

- Message Digest
- Id-contentType

- Id-messageDigest
- Id-aa-signingCertificateV2
- Id-aa-ets-sigPolicyId

A.1.5.4.3 Atributos Não-Assinados Obrigatórios

Não se aplica.

A.1.5.4.4 Referências à Cadeia de Certificação

O atributo Id-aa-signingCertificateV2 deve conter apenas referência ao certificado do signatário.

A.1.5.4.5 Valores da Cadeia de Certificação

Não se aplica.

A.1.5.4.6 Regras Adicionais do Signatário

A.1.5.5 Uso de Múltiplas Assinaturas

Na utilização de múltiplas assinaturas, todas elas devem empregar os mesmos algoritmos definidos no item A.1.9. As formas possíveis de múltiplas assinaturas é apenas:

- Co-assinaturas: quando a ordem de inserção das assinaturas não faz diferença;

A.1.5.5.1 Estruturas de Assinatura

No caso de co-assinaturas haverá múltiplas estruturas SignerInfo.

A.1.5.5.2 Conteúdo Dinâmico

O signatário é responsável por se certificar que o documento assinado não contém qualquer conteúdo dinâmico capaz de modificar o resultado do documento visualizado ao longo do tempo, como, por exemplo, quantias ou sentenças que se alteram após certa data.

A.1.5.6 Regras do Verificador

A.1.5.6.1 Atributos Não-Assinados Obrigatórios

Não se aplica.

A.1.5.6.2 Regras Adicionais do Verificador

Caso esteja presente mais de uma assinatura, aposta ao mesmo documento assinado, deve-se validar cada assinatura encontrada independentemente, segundo o item A.1.6.

A.1.6 Condições de Confiabilidade dos Certificados dos Signatários

A.1.6.1 Validação da Cadeia de Certificação

A.1.6.1.1 Raiz Confiável

A validação deve ser feita tomando como ponto de confiança o certificado da AC-Raiz UnB, disponível em https://www.unbsign/UnbSign-API/acraiz.

A.1.6.1.2 Restrição do Caminho de Certificação

O número máximo de certificados de AC, no caminho de certificação, entre o certificado do signatário e a AC-Raiz é 2 (dois).

A.1.6.1.3 Conjunto de Políticas de Certificado Aceitável

Assinaturas digitais geradas segundo esta Política de Assinatura deverão ser criadas com chave privada associada a certificado gerado pela API PKI de assinatura, conforme definido no DOC-ICP-04, com os seguintes OID:

- Tipo A1: OID 2.16.76.1.2.1.n
- Tipo A2: OID 2.16.76.1.2.2.n
- Tipo A3: OID 2.16.76.1.2.3.n
- Tipo A4: OID 2.16.76.1.2.4.n

A.1.6.1.4 Restrições de Nome

Não se aplica.

A.1.6.1.5 Restrições de Políticas de Certificado

Não se aplica.

A.1.6.2 Forma de Verificação do Status da Cadeia de Certificação (Revogação)

Tanto para o certificado do signatário quanto para os certificados das Autoridades Certificadoras da cadeia de certificação, a verificação do estado dos certificados deve ser realizada através de consulta à LCR (Lista de Certificados Revogados), usando os procedimentos definidos na RFC 3280, ou por meio de consulta OCSP (Online Certificate Status Protocol), usando os procedimentos definidos na RFC 2560.

A.1.7 Condições de Confiabilidade do Carimbo de Tempo

Não se aplica.

A.1.8 Condições de Confiabilidade dos Atributos

Não se aplica.

A.1.9 Conjunto de Restrições de Algoritmos

Para geração de assinaturas segundo esta política, podem ser utilizados os seguintes algoritmos:

- RSA/SHA256
- RSA/SHA-1