UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

SAMUEL DA SILVA

CRIPTOGRAFIA E SEGURANÇA APLICADA À INDÚSTRIA TÊXTIL: BLOCKCHAIN APLICADA NA AUTENTICAÇÃO DE DOCUMENTOS

Ouro Preto, MG
2025

SAMUEL DA SILVA

CRIPTOGRAFIA E SEGURANÇA APLICADA À INDÚSTRIA TÊXTIL: BLOCKCHAIN APLICADA NA AUTENTICAÇÃO DE DOCUMENTOS

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Carlos Frederico Marcelo da Cunha Cavalcanti

Ouro Preto, MG
2025

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

SAMUEL DA SILVA

Criptografia e Segurança aplicada à indústria têxtil: Blockchain aplicada na autenticação de documentos

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 08 de Setembro de 2025.

Membros da banca

Doutor Carlos Frederico Marcelo da Cunha Cavalcanti (Orientador) - Universidade Federal de Ouro Preto
Doutor Ricardo Augusto Rabelo Oliveira (Examinador) - Universidade Federal de Ouro Preto
Doutor Fernando Cortez Sica (Examinador) - Universidade Federal de Ouro Preto

Prof. Dr. Carlos Frederico Marcelo da Cunha Cavalcanti, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 28/10/2025.

Documento assinado eletronicamente por **Carlos Frederico Marcelo da Cunha Cavalcanti**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 18/12/2025, às 16:08, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **1034774** e o código CRC **AE2F9525**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Este trabalho é dedicado primeiramente à minha mãe Neide Xavier da Silva e meu pai Edson José da Silva que sempre me apoiaram. Em especial dedico a você Ane Karoline, que nunca deixemos de estar lado a lado.

Agradecimentos

A todos os professores do Departamento de Ciência da Computação, que sempre proporcionaram saber e conhecimento inigualáveis.

Aos meus amigos e colegas da academia, com quem compartilho objetivos e sonhos. Aos meus amigos de infância, que acompanharam, mutuamente, nosso desenvolvimento pessoal e as bifurcações dos caminhos

A todos os profissionais do Instituto de Ciências Exatas e Biológicas, que tornaram possível a existência de um ambiente de estudos excelente.

A cada escolha que fazemos, mil outras deixamos de fazer. A todos que ainda assim cruzaram o meu destino, sou grato por tê-los conhecido.

"Tudo o que temos de decidir é o que fazer com o tempo que nos é dado." (TOLKIEN, 1954, p. 52)

Resumo

A fim de buscar uma forma eficiente de gestão de transações digitais, esta pesquisa explora o uso da tecnologia *blockchain* e *Smart Contract* como soluções para garantir segurança e confiabilidade em ambientes *B2B (Business-to-Business)*. Com o objetivo de atender a esses requisitos, propõe-se a implementação de um sistema baseado em uma *blockchain* permissionada e privada, capaz de armazenar acordos firmados entre clientes e fornecedor, possibilitando auditorias detalhadas de requisições e assegurando registros imutáveis de transações realizadas. O estudo se concentra na aplicabilidade prática dessa tecnologia em contextos corporativos, destacando como os contratos inteligentes podem automatizar e reforçar a integridade de processos negociais. Além disso, a solução visa promover maior transparência nas relações comerciais, reduzir a necessidade de intermediários e mitigar riscos operacionais, contribuindo, para uma infraestrutura digital mais robusta e escalável.

Palavras-chave: *autenticação digital*, *blockchain*, *smart contract*.

Abstract

In order to pursue an efficient approach to managing digital transactions, this research explores the use of *blockchain* technology and *Smart Contracts* as solutions to ensure security and reliability in *B2B (Business-to-Business)* environments. To meet these requirements, the implementation of a system based on a private and permissioned *blockchain* is proposed, capable of storing agreements established between clients and suppliers, enabling detailed audits of requests and ensuring immutable records of completed transactions. The study focuses on the practical applicability of this technology in corporate contexts, highlighting how smart contracts can automate tasks and strengthen the integrity of business processes. Furthermore, the proposed solution aims to foster greater transparency in commercial relationships, reduce the need for intermediaries, and mitigate operational risks, thereby contributing to a more robust and scalable digital infrastructure.

Keywords: *digital authentication*, *blockchain*, *smart contract*.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API	Application Programming Interface
ERP	Enterprise Resource Planning
JSON	JavaScript Object Notation
PoS	Proof of Stake ou Prova de participação
PoW	Proof of Work ou Prova de trabalho
REST	Interface de programação de aplicações (API ou API web)
UFOP	Universidade Federal de Ouro Preto
HTTP	Hypertext Transfer Protocol
WSGI	Web Server Gateway Interface

Sumário

1 Introdução

O avanço tecnológico contínuo exige novas formas de autenticação, registro e armazenamento digital. Os métodos tradicionais, como registros físicos e autenticação cartorial, em uso no Brasil há mais de 460 anos (ANOREG-CE, 2022), têm se mostrado cada vez menos eficientes diante das demandas contemporâneas. Entre os principais problemas enfrentados, destacam-se o alto consumo de tempo de trabalho, vulnerabilidades relacionadas à espionagem industrial, facilidade de adulteração, extravio, deterioração e a constante necessidade de espaços físicos para armazenamento. Nesse cenário, tornam-se prioridades a eficiência e a segurança da informação, e a adoção de soluções digitais. Entretanto, a sofisticação das fraudes também evoluiu para atingir novos patamares.

Com a migração de operações contratuais e financeiras para o meio digital, diversas fraudes ocorrem ano após ano. De acordo com (CNN, 2024), um grupo criminoso aplicou um golpe milionário em Hong Kong por meio de falsificação da assinaturas de voz e imagem de membros da diretoria financeira da empresa vítima. Situações como essa reforçam a necessidade de novas políticas de segurança e sistemas de validação mais eficientes no meio empresarial.

Originalmente cunhado por Nick Szabo nos anos 90, o termo *smart contract* refere-se à automação de contratos legais. Atualmente, *smart contracts* também podem ser definidos como programas autoexecutáveis operando em uma *blockchain*, avaliando regras pré-definidas e realizando ações automáticas (*tasks*) caso condições sejam atingidas (ZOU et al., 2021).

Neste contexto, a tecnologia *blockchain*, também conhecida como livro-razão, surge como uma solução promissora para o problema, possibilitando a criação de bancos de dados transparentes e confiáveis na maioria das condições. Por meio de uma arquitetura distribuída e descentralizada, são garantidas a imutabilidade das informações, rastreamento, prevenção contra adulterações ou perdas de dados. A propagação de novos estados depende de um consenso entre nós participantes da rede *PoW*[1], *PoS*[2] ou de *nós avaliadores* em caso de redes permissionadas (PILKINGTON, 2016).

1.1 Justificativa

Erros, fraudes e falsificações em contratos geram prejuízos financeiros incalculáveis todos os anos, afetando tanto a credibilidade das partes quanto a estabilidade de processos internos. Nesse contexto, o reconhecimento digital legalmente válido das partes envolvidas mostra-se

[1] PoW *Proof of Work*: mecanismo de consenso utilizados em redes *blockchain* que valida transações por meio da resolução de problemas matemáticos complexos, exigindo alto poder computacional.

[2] PoS *Proof of Stake*: mecanismo de consenso utilizados em redes *blockchain* que seleciona validadores com base na quantidade de *tokens* em posse e em *stake*, em vez de poder computacional.

fundamental para garantir a segurança e a confiabilidade em transações contratuais.

Na indústria de manufatura, em especial, a adoção de métodos digitais de autenticação e validação de transações pode mitigar o risco de adulteração, extravio e retrabalho. Além disso, possibilita rastreabilidade mais eficiente e auditorias mais ágeis, assegurando transparência entre as partes. Assim, a autenticação digital de documentos de contratos de manufatura proposta neste trabalho visa resguardar os envolvidos e assegurar o cumprimento dos termos previamente assentidos.

Especificamente no segmento têxtil voltado à confecção de uniformes industriais, os desafios relacionados à transparência extrapolam as etapas de manufatura. A produção é destina ao uso pessoal de terceiros e as aquisições ocorrem por meio de intermediários, se estabelece um cenário em que a desconfiança entre as partes tende a ser a norma. Para mitigar ativamente esse ponto de ruptura, se torna necessária a adoção interfaces e órgãos mediadores imparciais.

1.2 Objetivos

A seguir são apresentados o objetivo geral e os específicos deste trabalho.

1.2.1 Objetivo geral

A pesquisa proposta tem como objetivo desenvolver um protótipo de aplicação validadora de transações cliente-servidor, utilizando *smart contracts* autoexecutáveis baseada em uma infraestrutura *blockchain* privada e permissionada, fundamentado em testes e na análise de dados coletados em uma indústria têxtil, Maximiano Uniformes e Confecções, na situada região dos Inconfidentes onde serão aplicados testes de funcionalidade através de abordagens qualitativas. O estudo busca garantir autenticidade e segurança em transações de equivalência contratual, valendo-se da tecnologia *blockchain* e integrando a *API REST* fornecida pelo sistema de gestão empresarial *NOMUS*, já adotado pela empresa.

1.2.2 Objetivos específicos

Pretende-se, primeiramente, identificar trabalhos correlatos e aplicações semelhantes, bem como definir os requisitos necessários à construção de uma arquitetura adequada. Em seguida, busca-se planejar a integração e a entrega contínua de funcionalidades, ao mesmo tempo em que se organizam e implementam as aplicações propostas. Para avaliar a eficácia do que foi desenvolvido, serão realizados testes de usabilidade, culminando na apresentação do protótipo.

1.3 Organização do trabalho

Este documento encontra-se estruturado em cinco capítulos. Esse capítulo introduz o trabalho situando o problema de pesquisa e descrevendo os objetivos. Em seguida, o Capítulo 2 reúne a revisão bibliográfica e o embasamento teórico, apresentando conceitos fundamentais. No Capítulo 3, aborda-se a metodologia empregada e os procedimentos de desenvolvimento. Os resultados e discussões são apresentados no Capítulo 4, contemplando análises e reflexões sobre o protótipo proposto. Por fim, no Capítulo 5, expõem-se as conclusões gerais do estudo e as possibilidades de trabalhos futuros.

2 Revisão Bibliográfica

Neste capítulo, são apresentados os trabalhos que fundamentam esta pesquisa, seguidos pelos conceitos teóricos que sustentam a arquitetura desenvolvida.

2.1 Trabalhos relacionados

A literatura sobre o uso de *blockchain* para validação de documentos e processos tem se expandido, com abordagens que variam em complexidade e escopo de aplicação. A análise desses trabalhos é fundamental para contextualizar as decisões de design e a contribuição específica desta monografia em relação ao estado-da-arte da tecnologia.

A necessidade de garantir a autenticidade de documentos digitais é um tema recorrente. Em (CHAUHAN et al., 2021), por exemplo, é proposto um *framework* genérico que explora o uso de funções *hash* e *blockchain* para facilitar o processo de autenticação, abordando o problema de forma mais ampla. Uma aplicação mais específica é apresentada em (SOMBRIO; ANTUNES; CASAGRANDE, 2024), onde é proposta uma solução para a autenticação de certificados e diplomas em instituições de ensino. Este trabalho utiliza a mesma plataforma de *blockchain* permissionada adotada aqui, a MultiChain, demonstrando sua viabilidade para criar registros imutáveis de documentos em cenários de baixa complexidade.

Em contraste, o estado da arte para cenários de alta complexidade, como as cadeias de suprimentos, aponta para soluções mais elaboradas. O *framework* proposto em (MANOHARAN, 2025) para a rastreabilidade de documentos na indústria química representa uma abordagem tecnologicamente mais avançada. Este trabalho utiliza a plataforma *Hyperledger Fabric*, um padrão para aplicações de *blockchain* empresarial, que oferece suporte nativo a *smart contracts* complexos e canais privados. Isso permite que a lógica de negócio seja executada de forma distribuída e validada pela própria rede (on-chain). Esta solução, foca em criar um sistema de rastreabilidade completo e *trustless*[1], onde a confiança é depositada no protocolo da *blockchain*, e não em um intermediário.

A principal contribuição deste trabalho não reside em um avanço no estado-da-arte da tecnologia *blockchain* em si, mas em seu valor como um estudo de caso sobre a aplicação específica. Ele oferece um modelo para empresas que buscam uma melhoria incrementar a segurança e auditabilidade de seus processos, transformando a *blockchain* em um uma fonte de registro que cria um dossiê digital auditável para cada transação, possibilitando a integração com sistemas de gestão centrais, sem a necessidade de uma substituição disruptiva.

[1] Trustless: Sistema onde os participantes não precisam confiar uns nos outros para validar transações.

2.2 Fundamentação Teórica

A construção de sistemas de software modernos, especialmente aqueles que gerenciam dados sensíveis, demanda a interação entre múltiplas camadas tecnológicas. A camada visível ao usuário, denominada *front-end*, é responsável pela interação direta e se manifesta, no contexto deste trabalho, como a interface de comunicação acessada pelo navegador. Ela engloba tanto a interface de usuário (UI) quanto a experiência de uso (UX), sendo implementada com tecnologias padrão da web: o HTML estrutura o conteúdo e os elementos visuais das páginas, como formulários e tabelas, definindo cada componente. Em complemento, o JavaScript adiciona interatividade e dinamismo, permitindo a captura de dados, validação de em tempo real e a comunicação assíncrona com o *back-end* para o envio e recebimento de informações sem a necessidade de recarregar a página (FREEMAN; ROBSON, 2020).

A segurança de sistemas digitais é fundamentada na criptografia, que provê mecanismos para assegurar a confidencialidade e a integridade dos dados. Para a confidencialidade, se destaca o uso de algoritmos de cifragem simétrica, como o AES *(Advanced Encryption Standard)*. O AES é um padrão de cifra de bloco adotado mundialmente que opera sobre blocos de dados de tamanho fixo e utiliza chaves de diferentes comprimentos para realizar múltiplas rodadas de substituição e permutação, garantindo um alto nível de segurança (STANDARDS; TECHNOLOGY, 2001). Já para a integridade e a autenticidade, o objetivo é a validação da origem e da inalterabilidade de documentos. Nesse contexto, a assinatura digital surge como uma solução robusta, vinculando criptograficamente a identidade de um autor a um documento por meio de algoritmos específicos (MONTEIRO; MIGNONI, 2007). Para alcançar essa garantia, a assinatura digital emprega um conjunto de ferramentas criptográficas, dentre as quais se destacam as funções de *hash*. Um algoritmo como a *hash* SHA-256, utilizado neste trabalho, processa uma entrada de dados para gerar uma "impressão digital"de tamanho fixo, que é então utilizada para garantir, de forma eficiente e segura, a integridade do documento original.

A execução dessas operações criptográficas e a gestão segura dos dados ocorrem na camada de *back-end*, que, em contrapartida ao *front-end*, constitui a parte não visível da aplicação. Essa camada arquitetural é responsável por gerenciar o processamento de informações, a lógica de negócio e a integração com bancos de dados (RICHARDS; FORD, 2020). A separação entre cliente *(front-end)* e servidor *(back-end)* é um princípio fundamental no design de arquiteturas de software distribuídas, pois permite a evolução independente dos componentes (FIELDING, 2000). A segurança e a confiabilidade das operações no *back-end* são cruciais, e é nessa camada que os princípios de criptografia são aplicados para proteger os dados processados. A aplicação desses mesmos conceitos em um ambiente de servidor distribuído é uma das bases da tecnologia *blockchain*, um sistema baseado em cadeias de blocos de informação interligados e descentralizados.

Também designada como "livro-razão distribuído", a *blockchain* é caracteriza por utilizar bases de registro e dados compartilhados entre múltiplos participantes (FERREIRA et al., 2017).

Sua estrutura é composta por cadeias de dados na qual nenhum bloco pode ser removido ou alterado sem invalidar toda a sequência subsequente. Para que um novo bloco seja adicionado, é necessário que um consenso seja alcançado entre os membros da rede denominados como nós. Todos os blocos são cifrados e convertidos em sequências *hash*, garantindo que os registros anteriores sempre representem um conjunto de dados válido e criptografado.

Para assegurar a integridade e a segurança na adição de novos blocos, as *blockchains* utilizam mecanismos de consenso. O mais conhecido é o *Proof of Work* (PoW), que se baseia na resolução de problemas matemáticos computacionalmente intensivos (NAKAMOTO, 2008). Em função de seu alto custo computacional, o PoW previne ataques de negação de serviço (DDoS), limitando o número de transações e garantindo segurança robusta. Como alternativa mais eficiente e sustentável, desenvolveu-se o *Proof of Stake* (PoS), que substitui o poder computacional pela participação financeira. Nesse modelo, os participantes da rede com maior quantidade de *tokens* (participação) possuem maior probabilidade de serem selecionados para validar um bloco, sendo implementados protocolos de aleatoriedade para evitar conluios (JUNIOR; DAHAB; HENRIQUES, 2023).

Sobre a infraestrutura da blockchain, é possível executar lógicas de negócio automatizadas por meio dos *smart contracts*. Originalmente definidos como um acordo mútuo com regras pré-estabelecidas, os *smart contracts* são programas de computador que se autoexecutam quando condições predefinidas são atendidas. Eles operam com base no princípio de "código como lei", no qual os termos de um acordo são traduzidos diretamente em código e armazenados na *blockchain*. Uma vez implantado, o *smart contract* torna-se imutável, e sua execução é distribuída e verificável por todos os participantes da rede, o que permite a validação de transações em ambientes sem confiança mútua *(trustless)*, eliminando a necessidade de uma autoridade externa (ZOU et al., 2021). Embora a execução da lógica *on-chain* (diretamente na blockchain) seja o paradigma ideal, a implementação prática em ambientes corporativos frequentemente adota uma abordagem híbrida. O conceito de *smart contract* é aplicado para descrever a lógica de negócio automatizada que opera *off-chain* (no servidor da aplicação), utilizando a *blockchain* como uma camada de registro imutável para ancorar e auditar os resultados dessas operações.

A Multichain é uma plataforma de código-aberto baseada na *blockchain* do Bitcoin (POLGE; ROBERT; TRAON, 2021). Ela oferece flexibilidade em configurações como disponibilidade de acesso, tamanho de bloco e recompensas de mineração. Sua API JSON-RPC permite que clientes, desenvolvidos em diversas linguagens, executem comandos como criação de *streams de dados*, publicação de blocos e consulta de informações. Essa versatilidade a torna viável para aplicações como rastreamento de cadeia de suprimentos e, em especial, autenticação de documentos.

Devido à necessidade de propagação e validação dos blocos de dados o uso de aplicações *blockchain* leva outro desafio, o armazenamento de arquivos de grande volume. Para

solucionar essa questão o IPFS *(InterPlanetary File System)*, um protocolo *peer-to-peer*[2] para armazenamento distribuído e descentralizado de arquivos se torna uma opção viável. Empregando endereçamento baseado em *hashes* criptográficas, o IPFS garante a integridade dos dados e oferece resiliência à censura e a falhas. Essa abordagem tem se mostrado eficiente para diminuir latências ao aproveitar *cachescaches*[3] locais, se tornando ideal para cenários corporativos que demandam alta disponibilidade e versionamento de documentos (BENET, 2014; ALI; KUMAR; SHARMA, 2020).

Para o desenvolvimento da camada de aplicação que possibilita a interação entre o usuário e infraestrutura distribuída o Flask, um *framework*[4] web em Python baseado no padrão WSGI, oferece um núcleo essencial para aplicações HTTP, permitindo a construção de sistemas modulares com APIs REST (GRINBERG, 2018). A organização do código em rotas denominadas *Blueprints* promove a modularidade e facilita a manutenção, possibilitando que diferentes partes da aplicação sejam desenvolvidas e testadas de forma isolada (RELAN, 2019). Essa camada também se integra a sistemas externos, como a ERP Nomus, um software de gestão empresarial focado em indústrias. Fundada em 2005, a ERP Nomus oferece uma plataforma web para planejamento e controle de processos produtivos, abrangendo desde controles financeiros à operações de *chão-de-fábrica* (PORTALERP, 2025). Sua API REST, autenticada por chave única, permite inserções e requisições em seu banco de dados utilizando o formato JSON (NOMUS, 2020; TECNOLOGIA, 2025). Através dessa integração novos usos para os dados armazenados são possibilitados.

Lançado em 2013, o Docker viabiliza a virtualização baseada em contêineres, que isolam processos e encapsulam aplicações em ambientes autossuficientes, viabilizando a consistência de ecossistema distribuídos. Ao compartilhar o *kernel*[5] do sistema operacional hospedeiro, os contêineres reduzem a sobrecarga de recursos e aceleram a inicialização (MERKEL, 2014). Por fim, para testes e avaliação de APIs REST, a ferramenta Insomnia possibilita simular requisições HTTP, inspecionar respostas de servidores, verificação de falhas, e realizar *benchmarks*, proporcionando agilidade na validação de integrações entre sistemas (INC., 2025).

[2] Peer-to-Peer: Arquitetura de rede em que cada nó funciona tanto como cliente quanto como servidor, permitindo a troca direta de dados sem a necessidade de um servidor central.

[3] Cache: Armazenamento temporário de dados que visa agilizar o acesso a informações já processadas ou solicitadas anteriormente.

[4] Framework: Conjunto de bibliotecas, ferramentas e boas práticas que fornece uma base estruturada para o desenvolvimento de aplicações.

[5] Kernel é o núcleo do sistema operacional, sendo responsável por gerenciar recursos e comunicação entre hardware e software.

3 Desenvolvimento

Este trabalho foi desenvolvido em duas partes. Na primeira, foi planejado o funcionamento e a arquitetura do servidor, bem como a interface de requisições. Na segunda, abordou-se a criação do ambiente de acesso para os clientes. Tais etapas são descritas ao longo deste capítulo, incluindo a metodologia empregada e as ferramentas necessárias.

3.1 Arquitetura

A arquitetura demonstrada na Figura 3.1, mostra a interação planejada dos usuários com o servidor de requisições.

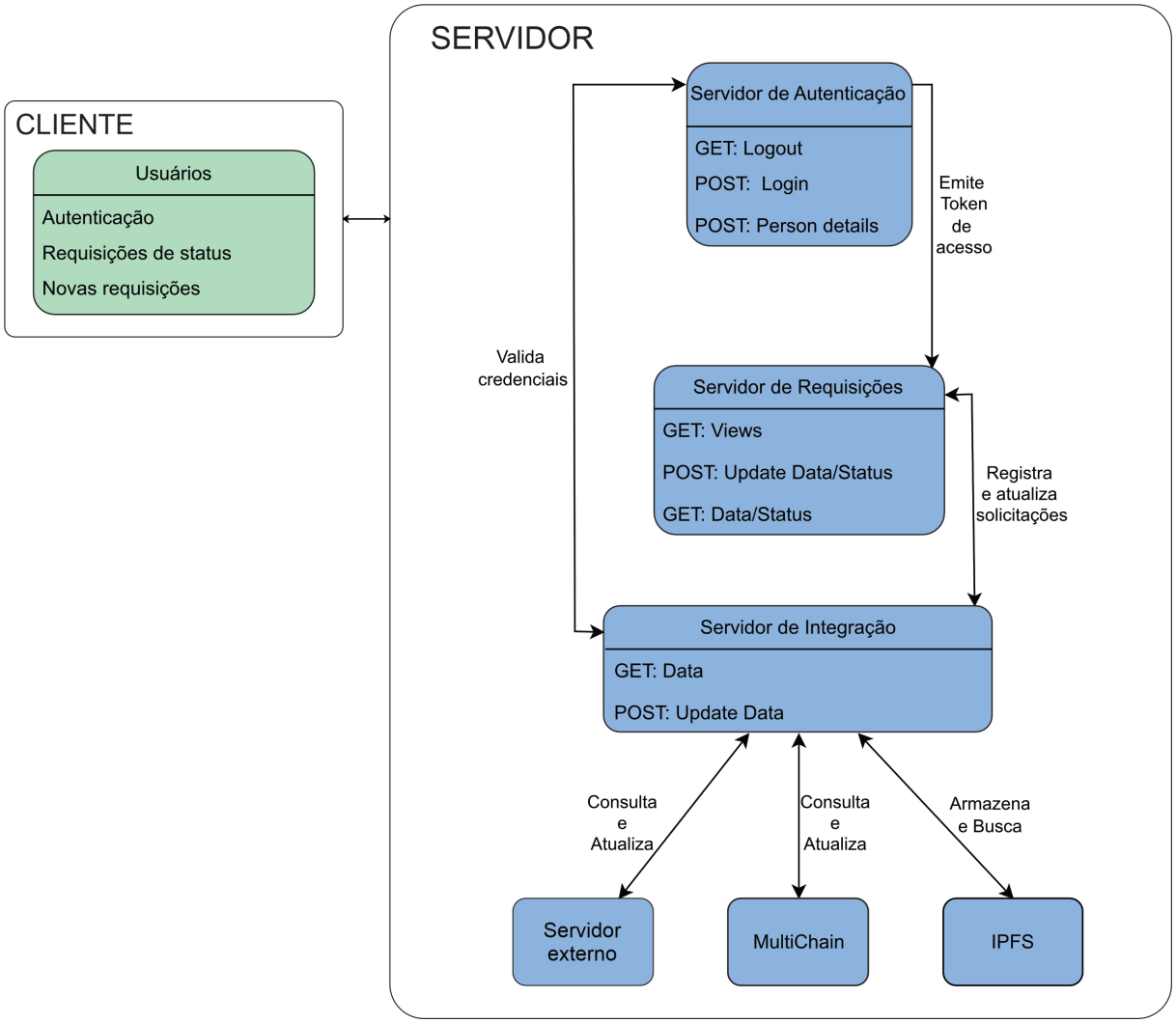


Figura 3.1 – Fluxo de requisições - Do próprio autor (2025)

O sistema é composto por dois grandes módulos: o *módulo Client* e o *módulo Server*. O *Client* submete requisições ao servidor e pode consultar requisições efetivadas, desde que sua

autenticação seja válida.

O módulo *server* é responsável por gerir usuários, conteúdos, acessos e restrições, devendo garantir que apenas usuários previamente autorizados realizem requisições. Ele é dividido em cinco submódulos:

- *Servidor de autenticação*: Responsável por verificar a autenticação do cliente, registrar eventos de acesso e gerenciar suas permissões.
- *Servidor de requisições*: Recebe requisições, executa regras definidas em *smart contracts*, e consultas a requisições anteriores para o *client*. Encaminha requisições confirmadas para o *Servidor de integração* e consulta seu status.
- *Servidor de integração*: Envia requisições recebidas, e verifica status de requisições ativas no módulo *Servidor externo ERP*.
- *Blockchain*: Armazena todos as requisições, atualizações, além das *hashes* contendo o caminho de acesso para dados armazenados de forma segura e imutável.
- *IPFS*: Armazena todos os arquivos registrados na blockchain.
- *Servidor externo ERP*: Representa um ambiente de integração de alto nível com o banco de dados externo utilizado pela *ERP Nomus*.

A implementação da lógica de negócio, foi aplicada ao diretamente ao servidor (off-chain). A principal razão para esta abordagem reside no não oferecimento de suporte nativo na MultiChain para a execução de smart contracts. Essa abordagem oferece gerenciamento ativo de dados e streams, controle granular de permissões e maior flexibilidade no desenvolvimento. Além disso, simplifica a integração com sistemas externos, como a API NOMUS empregada no servidor, garantindo uma comunicação mais robusta e eficiente.

3.1.1 Cenário de execução

Em um cenário de inserção de uma nova solicitação pelo *client*, o fluxo de execução será: Requisição de conexão pelo cliente, autenticação pelo servidor. Envio da nova solicitação ao servidor, recebimento pelo servidor de requisições. Execução do *smart contract* no servidor de requisições, envio de solicitação validada ao servidor de integração para o inserção de novo bloco de dados criptografado para a *blockchain/IPFS*. Inserção da solicitação no servidor externo ERP realizada pelo servidor de integração. Envio de informações de status ao servidor de integração pelo servidor externo ERP. Envio de informações de status ao cliente. Como pode ser visto na figura 3.2.

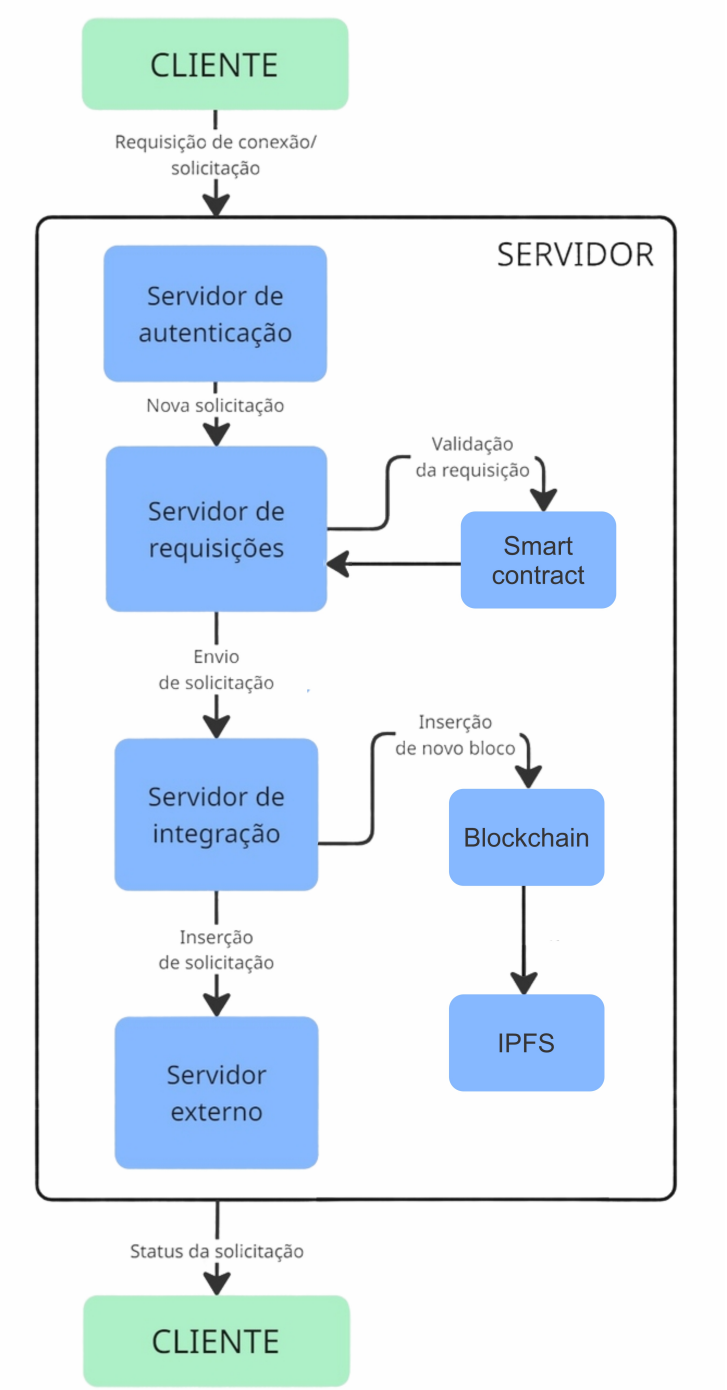


Figura 3.2 – Fluxograma de sucesso de inserção de solicitação - Do próprio autor (2025)

3.2 Especificação de requisitos

Esta seção tem como objetivo detalhar os requisitos do sistema, especificando as funcionalidades que precisam ser implementadas no software, bem como o comportamento esperado de cada uma.

3.2.1 Requisitos funcionais

Os requisitos funcionais descrevem especificações das funcionalidades de um sistema, detalhando o comportamento esperado de seus componentes.

- **Servidor de autenticação**: Deve ser capaz de permitir o login de usuários(cliente, financeiro ou entregador), registar eventos de autenticação, validar credenciais de representante, cliente e senha. Também deve gerar um *token* de acesso para requisições futuras, permitir que usuários encerrem sua sessão de forma segura e buscar informações de nome, CNPJ, e representante na API Nomus.
- **Servidor de requisições**: Deve ser capaz de receber e processar solicitações dos usuários, redirecionar usuários para páginas permitidas com base no seu perfil, enviar e receber status de solicitações para o Servidor de integração, atuando como um *proxy* seguro com o *frontend*.
- **Blockchain/IPFS**: Deve ser capaz de receber e armazenar solicitações de forma segura. Disponibilizar, listar, solicitações e atualizar solicitações já armazenadas.
- **Servidor de integração**: Deve ser capaz de processar a submissão de novas solicitações pelos usuários atualizando a *stream* correspondente na *blockchain*. Deve ser capaz de permitir a avaliação de solicitações efetuadas e exibir alertas de novas solicitações para o usuário correspondente, além de consultar e registar status de solicitações com o Servidor externo ERP.
- **Servidor externo ERP**: Deve ser capaz de integrar-se ao Servidor de integração. Também deve enviar status, e receber solicitações.

3.2.2 Requisitos não funcionais

Diferentemente dos requisitos funcionais, que focam no que deve ser feito, os requisitos não funcionais representam características e atributos de como as funcionalidades devem operar.

- **Servidor de autenticação**: Cada usuário deve possuir um identificador de autenticação e *id*[1] de acesso únicos. O servidor deve possuir disponibilidade de horários restrita, registro de *logs* para auditoria e rastreamento.
- **Servidor de requisições**: Para cada solicitação registrada, o usuário deve ter acesso as informações geradas, e status. Todo pedido recebido pelo Servidor de requisições deve ser avaliado utilizando regras de negócio relacionadas com o usuário (*smart contracts*).

[1] ID (*Identifier*): valor único atribuído a um elemento para identificá-lo dentro de um sistema ou base de dados.

- **Servidor de integração**: Toda solicitação recebida recebida deve ser válida, deve ser garantida a sincronização e consistência de dados entre o Servidor externo ERP e a *blockchain*. Deve consultar status de pedidos ativos e enviar para o servidor de requisições.
- **Blockchain/IPFS**: Todo conteúdo armazenado deve ser criptografado para garantir a segurança. A estrutura deve ser tolerante a falhas, assegurando a imutabilidade e integridade dos dados registrados.
- **Servidor externo ERP**: Para cada alteração de status o Servidor de integração deve ser comunicado. Falhas de comunicação devem ser informadas ao usuário.

3.3 Rotas

Cada módulo componente do servidor será uma *API REST* que responderá a chamadas usando protocolo HTTP. Através destas chamadas o sistema será integrado, acessando o conteúdo armazenado, e realizando consultas no Servidor externo ERP. Esta seção irá descrever as rotas de cada módulo do servidor.

3.3.1 Servidor de autenticação

Tabela 3.1 – Tabela de rotas do Servidor de Autenticação

Método	**Rota**	**Descrição**
POST	/auth/login	Esta rota envia as credenciais de autenticação do usuário, e emite um token de acesso
GET	/auth/logout	Esta rota encerra a sessão do usuário
POST	/api/person-details	Esta rota envia o ID do usuário para o servidor Externo e são retorna dados detalhados para o servidor de Autenticação

Fonte: Do próprio autor (2025)

3.3.2 Servidor de requisição

Tabela 3.2 – Tabela de rotas do Servidor de requisição

Método	**Rota**	**Descrição**
GET	/home	Esta rota autentica e redireciona o usuário para sua página inicial correspondente ao seu perfil (cliente, financeiro, entregador)

Fonte: Do próprio autor (2025)

Tabela 3.3 – Tabela de rotas do Servidor de requisição - Cliente (views)

Método	**Rota**	**Descrição**
GET	/dashboard/cliente	Esta rota exibe a página inicial para o usuário cliente
GET	/orders/new	Esta rota exibe a página de criação e submissão de novos pedidos
GET	/orders/finished	Esta rota exibe o histórico de status e pedidos para o usuário cliente
GET	/finished/deliveries	Esta rota exibe o histórico de entregas finalizadas e aprovadas para o usuário cliente
GET	/contract	Esta rota exibide a página de visualização dos detalhes do contrato mestre e *iframe* de controle de peças para o usuário cliente

Fonte: Do próprio autor (2025)

Tabela 3.4 – Tabela de rotas do Servidor de requisição - Cliente (proxy)

Método	**Rota**	**Descrição**
POST	/api-proxy/order/submit	Esta rota encaminha a requisição de novo pedido para o *integration_server*
GET	/api-proxy/contract/status	Esta rota busca o status atual do contrato e o saldo de produtos na *blockchain*

Fonte: Do próprio autor (2025)

Tabela 3.5 – Tabela de rotas do Servidor de requisição - Financeiro (views)

Método	**Rota**	**Descrição**
GET	/dashboard/financeiro	Esta rota exibe a página principal para o usuário financeiro
GET	/orders/requests	Esta rota exibe a página contendo a lista de pedidos aguardando aprovação
GET	/confirmation	Esta rota exibe a página contendo a lista de romaneios de entrega aguardando aprovação
GET	/warnings	Esta rota exibe a página contendo a lista de alertas e notificações de solicitação

Fonte: Do próprio autor (2025)

Tabela 3.6 – Tabela de rotas do Servidor de requisição - Financeiro (proxy)

Método	**Rota**	**Descrição**
GET	/api-proxy/orders/list	Esta rota obtém a lista de pedidos pendentes de avaliação do *integration_server*
POST	/api-proxy/order/review	Esta rota envia a decisão de aprovação de um pedido para o *integration_server*
GET	/api-proxy/orders /view/<ipfs_hash>	Esta rota retorna o PDF de um pedido para ser visualizado no modal
GET	/api-proxy /deliveries/pending-approval	Esta rota obtém a lista de entregas que aguardam aprovação do *integration_server*
POST	/api-proxy/delivery/approve	Esta rota envia a aprovação de uma entrega para o *integration_server*
GET	/api-proxy/deliveries /view/<ipfs_hash>	Esta rota retorna o PDF de uma prova de entrega para ser visualizado
GET	/api-proxy /notifications/consolidated	Esta rota obtém as lista de alertas e notificações consolidados de várias fontes

Fonte: Do próprio autor (2025)

Tabela 3.7 – Tabela de rotas do Servidor de requisição - Entregador (views)

Método	**Rota**	**Descrição**
GET	/dashboard/entregador	Esta rota exibe a página principal para o usuário entregador
GET	/deliveries	Esta rota exibe a página de lista romaneios aguardando a submissão da prova de entrega

Fonte: Do próprio autor (2025)

Tabela 3.8 – Tabela de rotas do Servidor de requisição - Entregador (proxy)

Método	**Rota**	**Descrição**
GET	/api-proxy/deliveries /entregador	Esta rota obtém a lista de romaneios disponíveis do *integration_server*.
POST	/api-proxy/delivery/submit	Esta rota envia a prova de entrega (PDF/imagem) para o *integration_server*, que processa o arquivo e atualiza a blockchain.

Fonte: Do próprio autor (2025)

Tabela 3.9 – Tabela de rotas do Servidor Integração - Apoio

Método	**Rota**	**Descrição**
POST	/api/person-details	Esta rota busca informações de uma pessoa (titular do contrato, representante) no Servidor Externo, durante o login
GET	/api/contract/status	Esta rota retorna metadados do contrato e status atual do inventário de produtos na *blockchain*
GET	/api/contract/view	Esta rota retorna o PDF do contrato mestre descriptografado
GET	/api/notifications /consolidated	Esta rota busca e consolida alertas de todas as *streams* armazenadas na *blockchain*
GET	/api/notifications/list	Esta rota retorna a lista detalhada do registro de notificações

Fonte: Do próprio autor (2025)

Tabela 3.10 – Tabela de rotas do Servidor integração - Fluxo de pedidos

Método	**Rota**	**Descrição**
POST	/api/order/submit	Esta rota processa a submissão de um novo pedido, gera PDF, realiza assinatura digital, criptografa, envia ao IPFS e registra hash na *blockchain*
GET	/api/orders/list	Esta rota retorna a lista completa de pedidos com status mais recente
POST	/api/order/review	Esta rota recebe decisões de aprovação do usuário financeiro, atualiza status na *blockchain* e cria notificação na pagina principal para o usuário cliente
GET	/api/orders/view <ipfs_hash>	Esta rota retorna PDF de um pedido específico após descriptografia

Fonte: Do próprio autor (2025)

3.3.3 Servidor de integração

Tabela 3.11 – Tabela de rotas do Servidor integração - Fluxo de entregas

Método	Rota	Descrição
GET	/api/deliveries/entregador	Esta rota retorna a lista de romaneios disponíveis para a página de submissão para o usuário entregador
POST	/api/delivery/submit	Esta rota processa submissão de prova de entrega (PDF ou imagem), converte se necessário, adiciona assinatura, envia ao IPFS e atualiza status na *blockchain*
GET	/api/deliveries/pending-approval	Esta rota retorna romaneios já submetidos para o usuário financeiro
POST	/api/delivery/approve	Esta rota processa a aprovação final de entrega para o usuário financeiro, e atualiza o status na *blockchain*
GET	/api/deliveries/list	Esta rota retorna a lista consolidada de entregas aprovadas para a página correspondente do usuário cliente e financeiro
GET	/api/deliveries/ view/<ipfs_hash>	Esta rota retorna o PDF de uma prova de entrega a partir do IPFS

Fonte: Do próprio autor (2025)

3.4 Implementação

Nesta seção, são apresentados trechos e detalhamentos da implementação das principais rotas e funcionalidades da aplicação proposta.

- *Inicialização do ambiente:* A base para a operação da aplicação proposta reside em uma configuração composta por três pilares: a blockchain *Multichain*, o sistema de arquivos distribuído *IPFS* e o servidor *Flask* ambos inicializados em contêineres através do *Docker*. A figura A.1 mostra a inicialização da blockchain, logo em seguida os nós em secundários também são iniciados A.2 usando contêinere individuais para a construção da rede denominada *nomus_chain*. Logo em seguida, como pode ser visto em A.3, as *streams* pertencentes à blockchain são criadas e iniciadas pelo contêiner auxiliar de *setup*, garantindo a integridade e rastreabilidade das informações além de separação de dados categorias.

3.4.1 Servidor de autenticação

O servidor de autenticação é responsável por validar a identidade dos usuários e gerenciar suas permissões e sessão de forma segura controlando o acesso à aplicação.

- **Autenticação de usuário:** O trecho de código da Figura B.1 mostra a como é feita a autenticação do usuário. O processo de autenticação é iniciado quando são submetidas

as credenciais (*login, representante, e senha*) através da rota *login*. O sistema faz uma pré carga de usuários autorizados e a senha fornecida pelo usuário é convertida para *hash SHA256* e comparada. Caso os dados fornecidos sejam inválidos, uma mensagem de erro é exibida. Caso a autenticação seja bem-sucedida, o servidor de autenticação realizada uma chamada ao *Servidor externo* para consultar, em tempo real, os dados cadastrais correspondentes a *API da ERP Nomus* representada pelo Servidor Externo.

Com os dados validados uma sessão segura é criada, armazenando informações cruciais(*user_id*, *user_role*) e um *token* de acesso é emitido e o usuário é redirecionado para sua respectiva página inicial.

3.4.2 Servidor de integração

Este módulo é responsável pelo núcleo de lógica da aplicação, possibilitando interações com a *blockchain*, o IPFS e o Servidor Externo.

- **Criptografia:** Para garantir a segurança de arquivos com conteúdo sensível foi usada o a biblioteca "Fernet", que combina o algoritmo AES com HMAC usando *SHA256* para a autenticação da mensagem. Na figura B.2 está descrito o trecho de código usado para criptografar, seguido do trecho de código para descriptografar um arquivo.

- **Visualização de documentos:** Para que todos os documentos enviados para a *blockchain* possam ser exibidos de maneira segura o servidor utiliza a hash fornecida para requisitar o arquivo criptografado na IPFS. A chave de descriptografia é então utilizada para decifrar o conteúdo e transmiti-lo para o usuário sem realizar alterações, garantindo a integridade e confiabilidade das informações. Conforme mostrado na Figura B.3.

- **Submissão de pedido:** Para que um pedido possa ser enviado para o servidor o primeiro passo é a atualização do inventário. O servidor consulta a *stream inventory_stream* da *blockchain*, incrementa o campo *consumed_stock* com a quantidade solicitada e publica o novo registro. Em seguida é gerado um documento PDF, capturando as informações exibidas no *frontend* juntamente uma assinatura capturada através da biblioteca *signaturePad*. Este documento é então criptografado e inserido na IPFS. Finalmente, um registro completo do pedido é criado em formato JSON, e publicado na *orders_stream* como pode ser visto em B.4.

- **Submissão de comprovante de entrega:** Para lidar com eventos de confirmação de entrega, o servidor solicita ao usuário com perfil *entregador* o envio de um comprovante(foto ou PDF) através da interface. O servidor então, criptografa e envia o novo arquivo para a IPFS, além de publicar o hash obtido na *blockchain*. B.5.

- **Aprovação de pedidos e comprovantes de entrega:** Pedidos e comprovantes submetidos são listados para o usuário com perfil *financeiro* para que possa ser tomada decisão de

aceite ou recusa, sendo uma forma de *three-way handshake*[2]. O servidor localiza na *stream* correspondente na *blockchain* o comprovante de entrega avaliado, e como poder visto na Figura B.7, e atualiza seu *status*.

Já os registros de decisão de aprovação de pedidos são acionados pela rota vista na Figura B.6. Seu funcionamento é semelhante ao descrito anteriormente para a avaliação de entregas. Adicionalmente, uma notificação da decisão é publicada na *blockchain*, para consulta e registro de eventos.

3.4.3 Servidor de requisição

Para que seja possível a integração do *frontend* com o *backend*, o servidor de requisições realiza dupla função, atuando como *proxy* seguro para chamadas ao servidor de integração e controlando o acesso às páginas de navegação. O *token* de acesso fornecido pelo servidor de autenticação é usando para restringir acesso a fluxos de execução de outros tipos de perfis C.1. Quando uma ação que necessita de dados fornecidos pelo *backend* é realizada, o código JavaScript na página correspondente realiza uma requisição *proxy* ao servidor de requisição que então repassa a requisição para o servidor de integração C.2.

- **Blockchain:** A blockchain, como abordado anteriormente, é uma estrutura de dados que garante a integridade e imutabilidade dos dados armazenados. Na Figura C.3. são mostrados os métodos de inserção e recuperação de dados na Multichain, com destaque para o componente *get_stream_item* que busca o item correspondente no *data_stream*. Já as inserções de conteúdo são feitas pela chamada *publish* na API JSON-RPC da Multichain, especificando a *stream* destinada C.4.
- **Servidor Externo:** Cada tipo de dado utilizado pela aplicação possui uma chamada *GET/POST* diferente na Api Nomus, sendo retornados em ambos os casos em formato *JSON*. Em C.5. é realizada uma requisição proxy pelo servidor de requisições, onde podemos verificar também seu retorno.

[2] *three-way handshake*: sequência de três mensagens trocadas entre cliente e servidor para estabelecer uma conexão confiável.

4 Resultados

Este capítulo apresenta a validação empírica do protótipo desenvolvido, detalhando os testes realizados para aferir a funcionalidade, a robustez e o desempenho da arquitetura proposta.

A metodologia de validação foi estruturada em três etapas principais: primeiramente, a verificação da infraestrutura de contêineres. Em segundo lugar, a validação funcional do fluxo de operações através de testes de API que simulam um ciclo de básico de vida do usuário. E por fim, a análise de desempenho, que inclui testes de carga nos componentes individuais e um *benchmark* comparativo com uma arquitetura de banco de dados tradicional. Os testes foram executados utilizando a ferramenta Insomnia para a simulação de requisições HTTP e a inspeção das respostas do servidor.

4.1 Validação da Infraestrutura

Antes de validar a aplicação, é fundamental garantir que a infraestrutura subjacente, definida pelo Docker Compose, esteja operacional. A execução do comando *docker-compose up* confirmou que todos os nove contêineres definidos (blockchain, node-1, node-2, node-3, node-4, node-5, ipfs, initializer, dapp) foram iniciados com sucesso e se encontravam em estado de execução (Up), com exceção do contêiner *initializer* utilizado para criar as *streams* e popular a *blockchain* com os dados iniciais. Análises dos logs de cada contêiner não revelaram erros críticos, confirmando a saúde e a estabilidade do ambiente de implantação como pode ser visto na Figura 4.1.

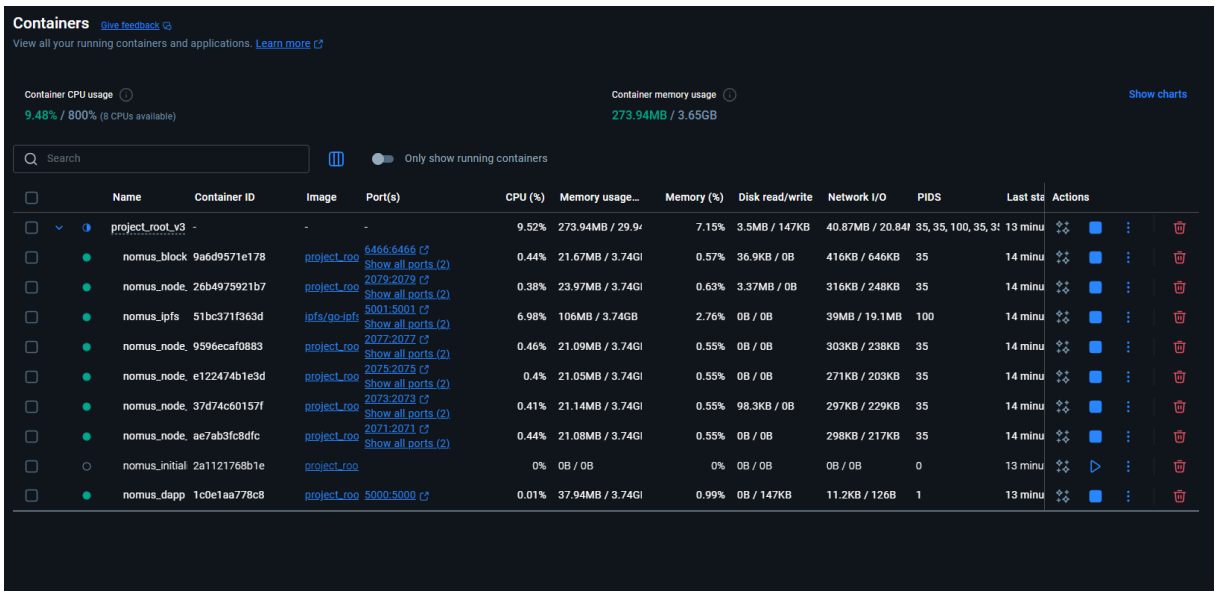


Figura 4.1 – Contêineres docker em execução - Do próprio autor (2025)

4.2 Ciclo de Vida de um Pedido

- **Submissão de Pedido:** Com um *token* de acesso válido para um usuário com perfil de "cliente", uma requisição POST para a rota */order/submit*, contendo os itens do pedido e a imagem da assinatura, foi processada com sucesso. O sistema retornou o documento PDF gerado e assinado, confirmando que os dados foram enviados ao IPFS e o registro foi armazenado na *blockchain* como pode ser visto na Figura 4.2.

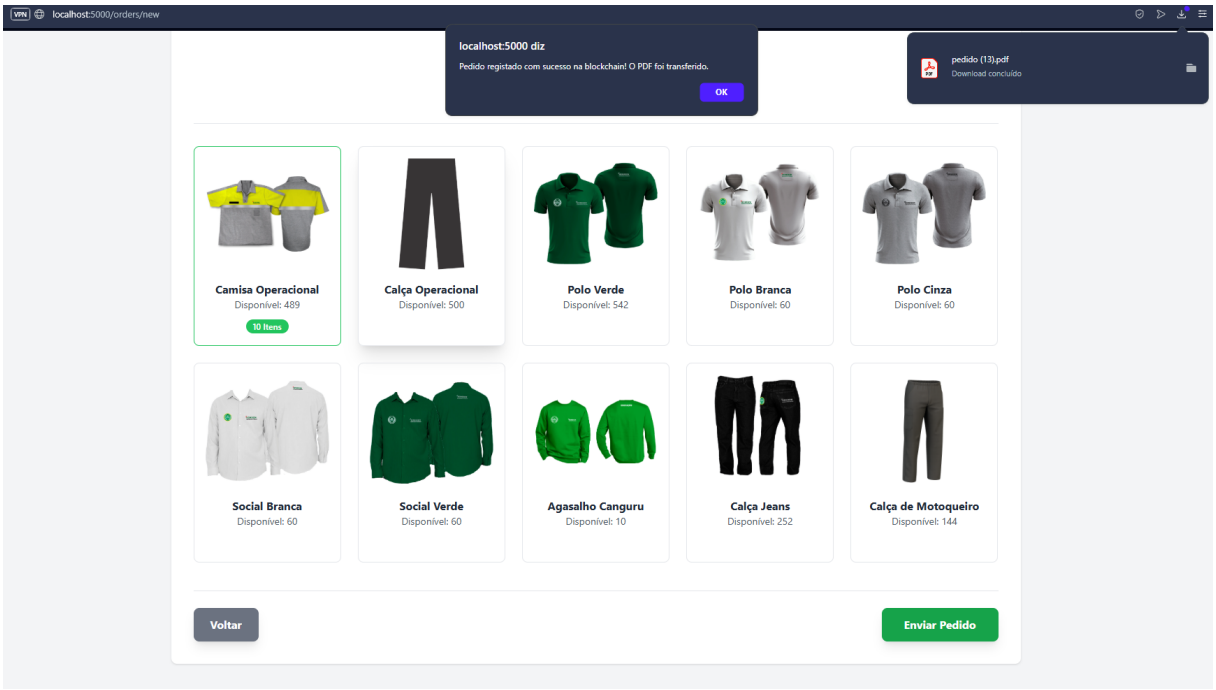


Figura 4.2 – Página de envio de pedidos - Do próprio autor (2025)

- **Revisão e Aprovação:** Utilizando um *token* de um usuário "financeiro", uma requisição POST para a rota */order/review* com a decisão "approved"foi enviada, como pode ser visto na Figura 4.3. O sistema respondeu com sucesso, indicando que o status do pedido foi atualizado na *blockchain* e uma notificação foi gerada.

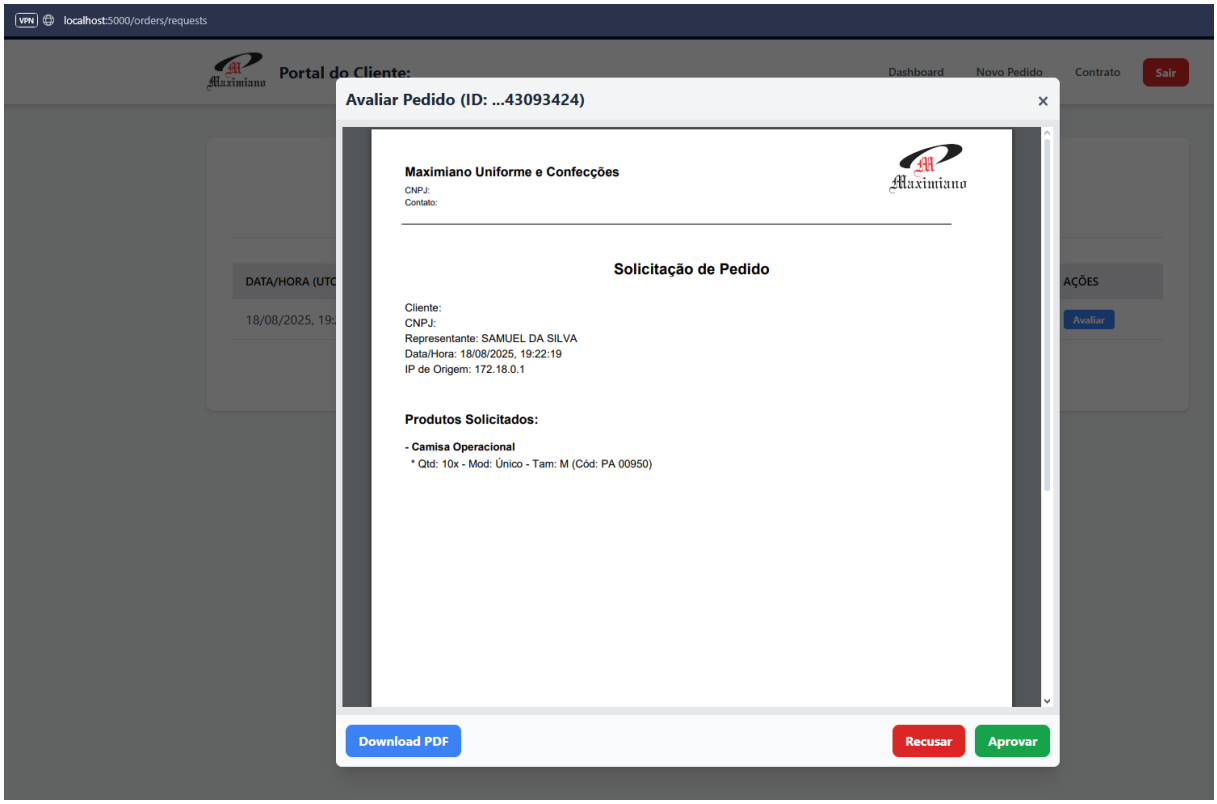


Figura 4.3 – Página de revisão de pedidos - Do próprio autor (2025)

- **Submissão de Comprovante:** Seguindo uma lógica similar, com um *token* de usuário com perfil "entregador", uma requisição POST para a rota */delivery/submit*, contendo a chave da entrega e o arquivo de comprovante (PDF, JPG, PNG) como pode ser visto na Figura 4.4, foi bem-sucedida, retornando o *txid* da transação na *blockchain*.

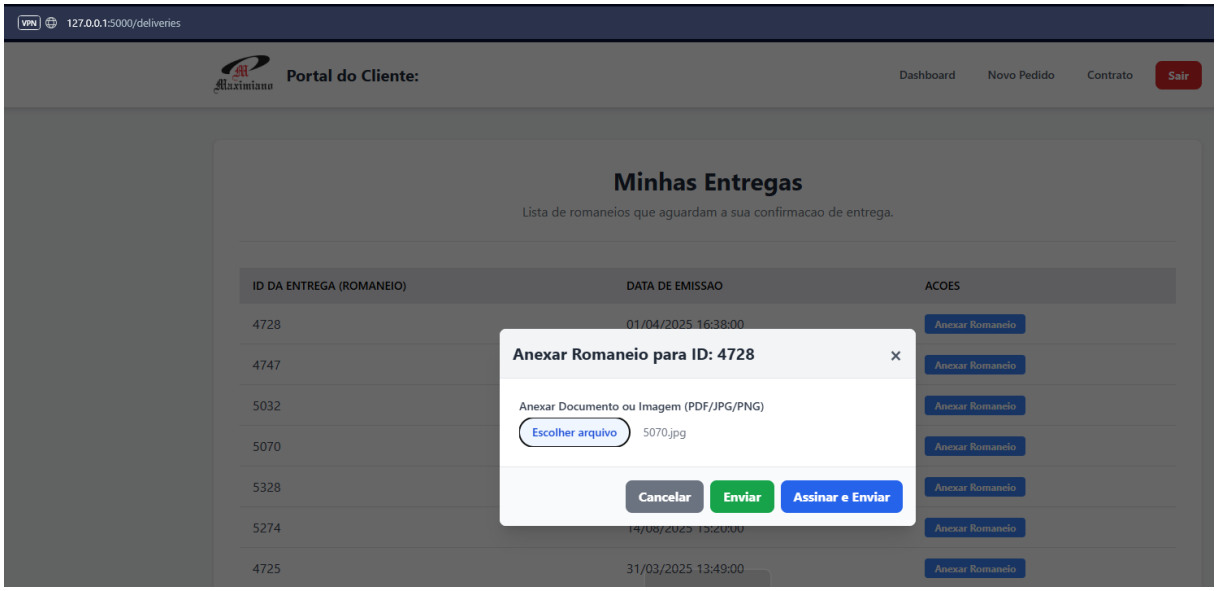


Figura 4.4 – Página de envio de comprovantes de entrega - Do próprio autor (2025)

Esta abordagem de testes em fluxo demonstrou que o sistema não apenas executa funções

individuais, mas também mantém o estado e a lógica de negócio de forma coesa através das diferentes etapas do processo.

4.3 Testes de rotas e Benchmark de desempenho

Para contextualizar os resultados de desempenho e justificar a escolha da arquitetura, foram realizados testes de estresse e resposta nas rotas da aplicação e conduzido um *benchmark* comparativo entre a implementação atual *blockchain* e um banco de dados relacional para registro de transações.

Os testes de estresse consistiram na execução de 100 transações para cada rota, utilizando o executor de testes da ferramenta Insomnia. Os dados obtidos podem ser vistos a seguir.

- **Servidor de autenticação**

  Autenticação de usuário :

  A autenticação dos usuários previamente registrados, pode ser visto na Figura 4.5. Após o sucesso do login o *token* de casso é armazenado na sessão do navegador.

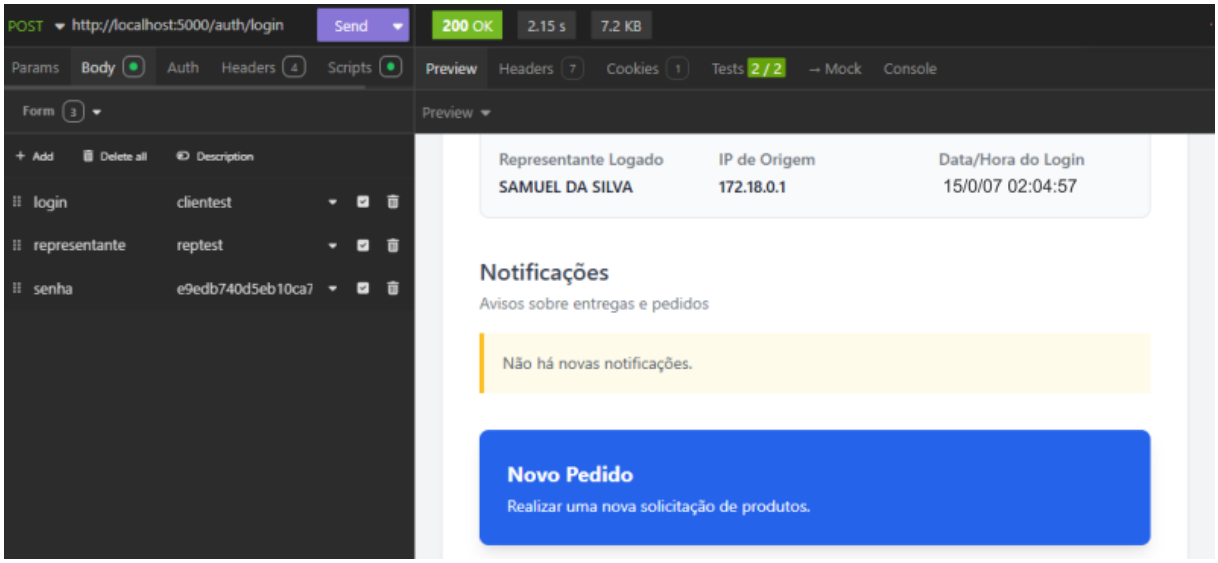


Figura 4.5 – Teste da rota /auth - Do próprio autor (2025)

A Figura 4.6 mostra a falha de autenticação gerada por um usuário não registrado.

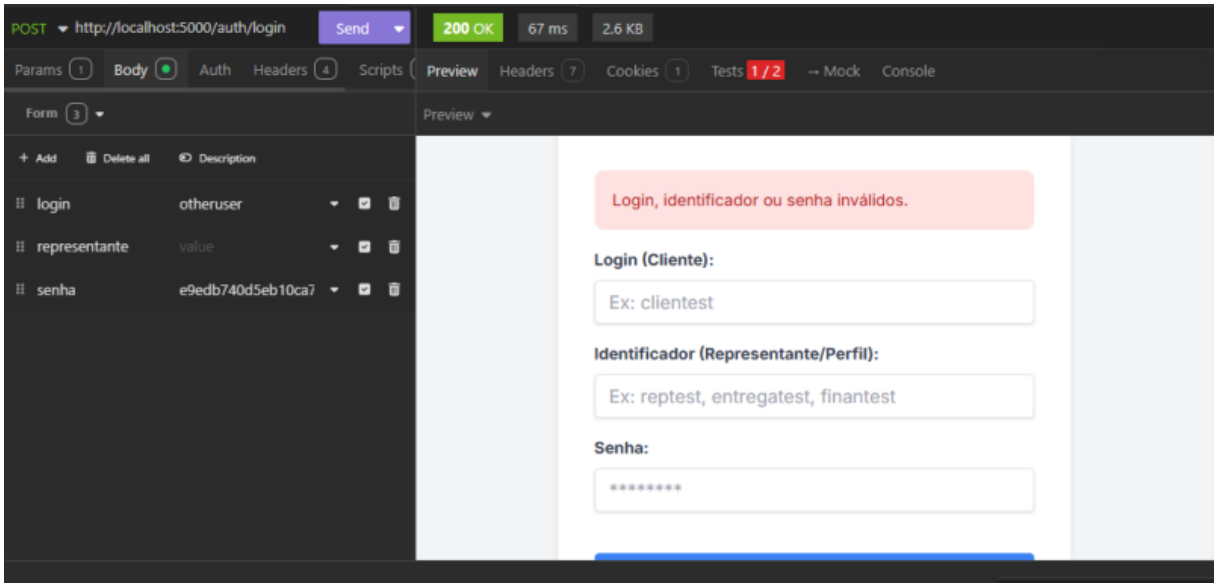


Figura 4.6 – Teste da rota /auth - Do próprio autor (2025)

- **Servidor de requisição**

  Submissão de pedido: A Figura 4.7 mostra os arquivos de pedido enviados ao servidor, onde o *token* do tipo especifico de usuário (cliente) é requerido.

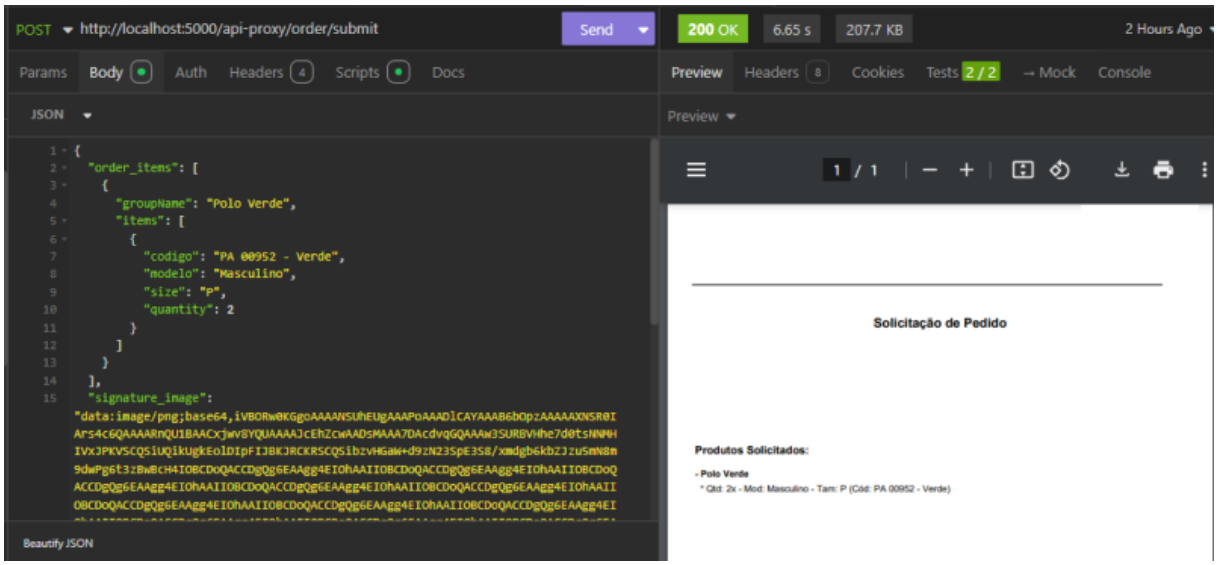


Figura 4.7 – Teste da rota /order/submit - Do próprio autor (2025)

O teste onde o *token* é inválido é apresentado na Figura 4.8 onde temos um erro de autenticação.

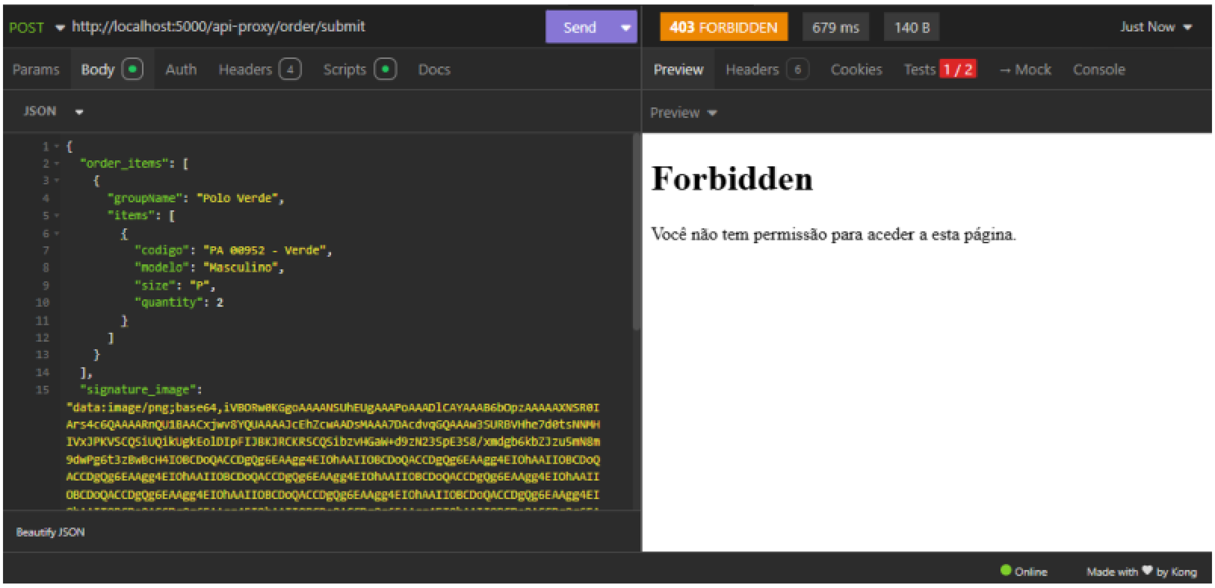
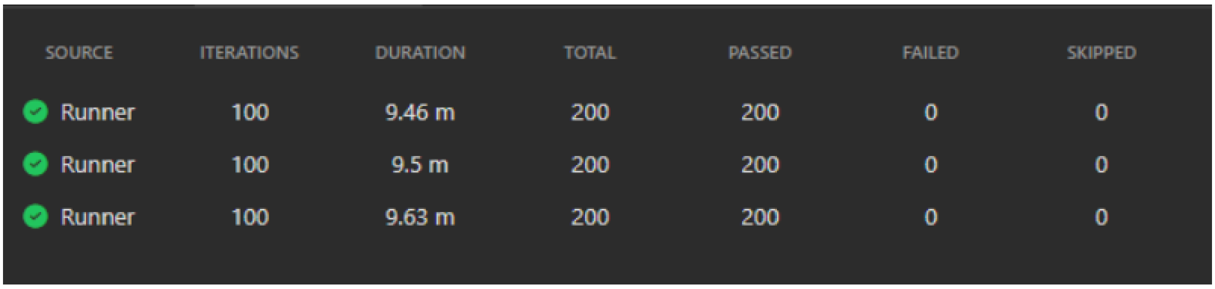


Figura 4.8 – Teste da rota /order/submit - Do próprio autor (2025)

Na Figura 4.9, se pode ver os testes de estressamento realizados.



SOURCE	ITERATIONS	DURATION	TOTAL	PASSED	FAILED	SKIPPED
Runner	100	9.46 m	200	200	0	0
Runner	100	9.5 m	200	200	0	0
Runner	100	9.63 m	200	200	0	0

Figura 4.9 – Teste da rota /order/submit - Do próprio autor (2025)

Submissão de comprovante de entrega: Na Figura 4.10 podemos observar à anexação e envio de comprovantes de entrega. Também é necessário a validação do *token* do usuário(entregador).

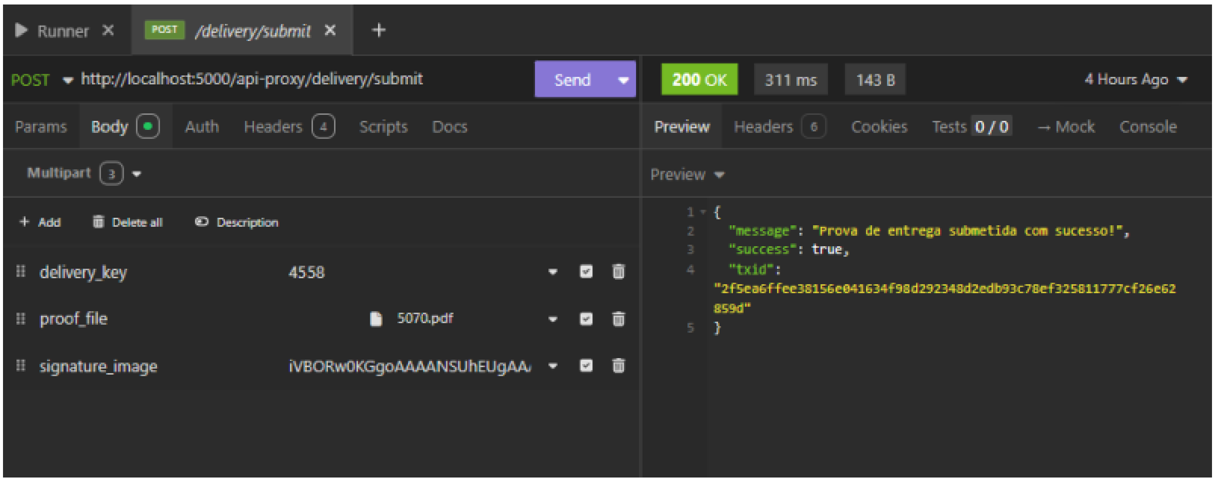


Figura 4.10 – Teste da rota /delivery/submit - Do próprio autor (2025)

O teste onde o *token* é inválido é apresentado na Figura 4.11 onde temos um erro de autenticação.

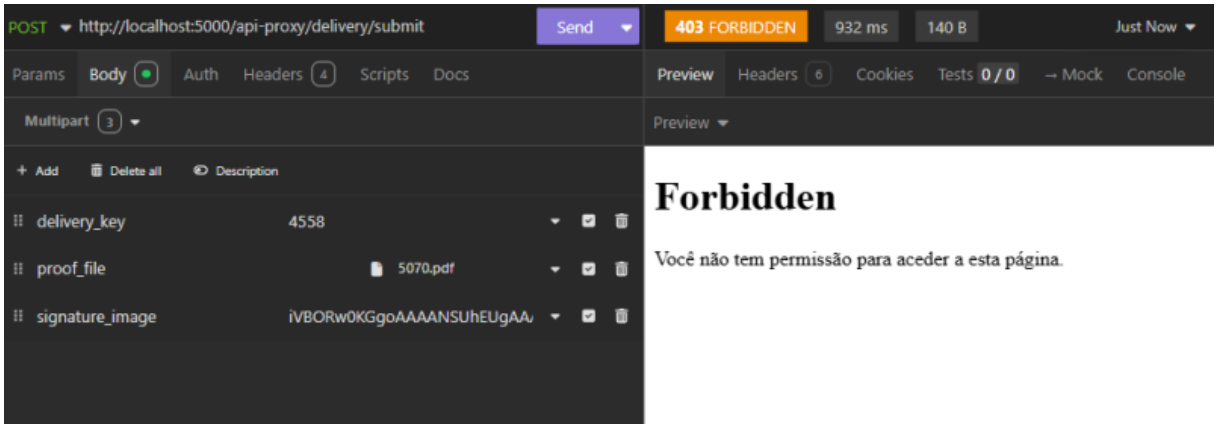


Figura 4.11 – Teste da rota /delivery/submit - Do próprio autor (2025)

Aprovação de pedidos e comprovantes de entrega: A Figura 4.12 mostra a requisição enviada através da decisão tomada (*Aprovar pedido*).

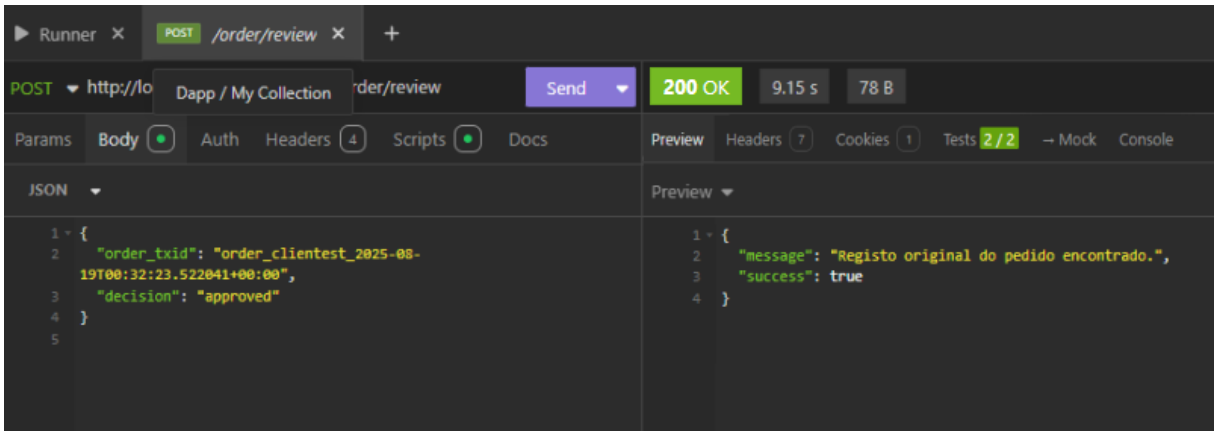


Figura 4.12 – Teste da rota /order/review - Do próprio autor (2025)

Já na Figura 4.13 podemos observar a requisição enviada após decisão *(Confirmar entrega)*.

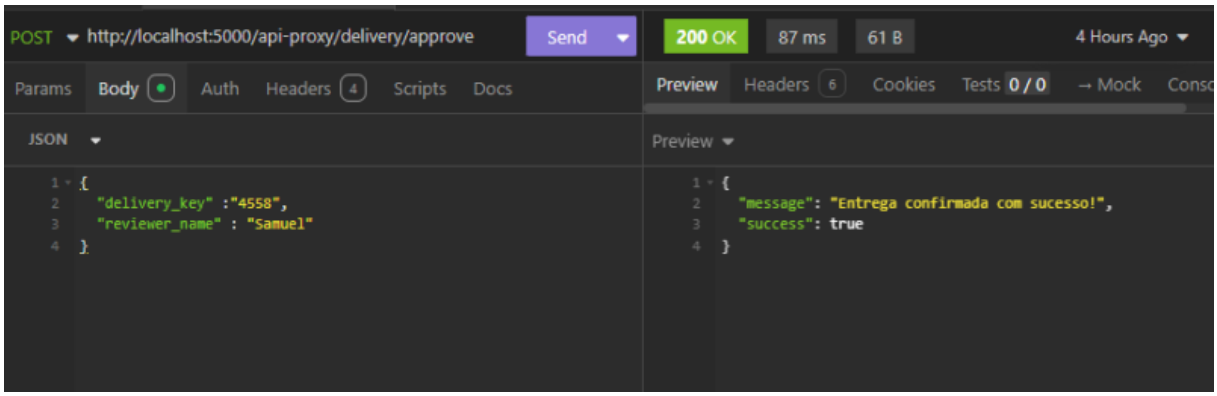


Figura 4.13 – Teste da rota /delivery/approve - Do próprio autor (2025)

Ambas as rotas anteriores tem funcionamento parecido, sendo alterado principalmente o tipo de usuário provedor dos dados avaliados.

- **Servidor de integração**

  Blockchain:

  Os testes de requisição na *blockchain*, como pode ser visto na figura 4.14, indicam o total porcentagem de sucesso nas operações realizadas. O tempo médio de inserção dos dados foi relativamente baixo e apresentando pouca variação com o aumento da latência máxima.

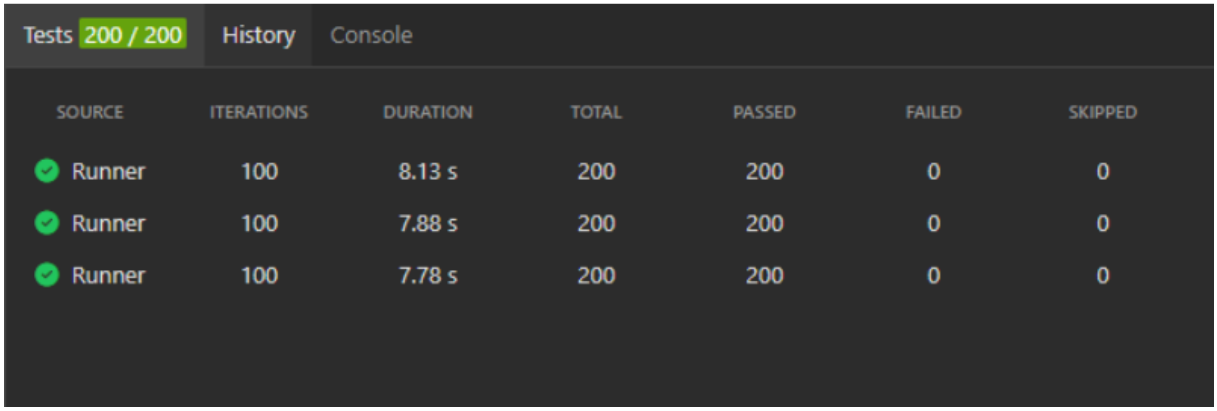


Figura 4.14 – Teste de estresse/carga na *blockchain* - Do próprio autor (2025)

IPFS: Os testes de requisição no IPFS, podem ser vistos na Figura 4.15. Sendo contabilizado o tempo total necessário para a disponibilização de um arquivo PDF através de sua *SHA256* previamente obtida na *blockchain*, seguido de sua descriptografia.

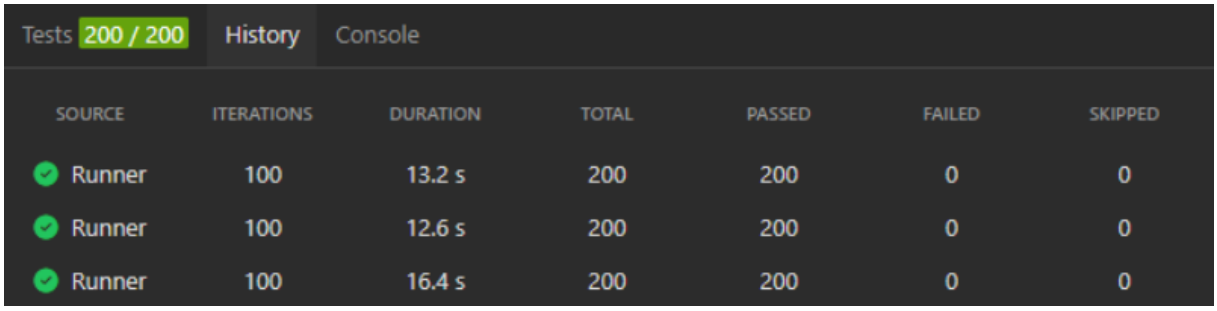


Figura 4.15 – Teste de estresse/carga no IPFS - Do próprio autor (2025)

Servir externo: Os testes de requisição ao Servidor Externo, podem ser vistos na Figura 4.16. Diferentemente dos outros contêineres, as requisições a API NOMUS exigem um intervalo de tempo (delay) elevado entre as requisições.

Figura 4.16 – Teste de estresse/carga no Servidor externo - Do próprio autor (2025)

Os testes de estresse, apresentados nesta seção, foram analisados para extrair métricas quantitativas de desempenho para cada componente principal da arquitetura.

Tabela 4.1 – Benckmark comparativo de componentes

Componente	**Operação**	**Duração Média (s)**	**Vazão Média (TPS)**	**Latência Média (ms)**
MultiChain	Escrita e Leitura	7,93	25,22	39,65
IPFS	Recuperação e Decriptografia	14,07	14,21	70,36
Servidor Externo (API NOMUS)	Consulta de Dados	138,40	0,60	1384,00

Fonte: Do próprio autor (2025)

O resultado mais significativo, como pode ser visto na Tabela 4.1 é a identificação do Servidor Externo (API NOMUS) como o principal gargalo de desempenho e estabilidade do sistema. Com uma vazão de apenas 0.60 TPS e uma latência média superior a 1.3 segundos, os testes também revelaram uma taxa de falha significativa (16.7% em 300 iterações) sob carga sustentada ao se diminuir o tempo máximo de resposta tolerado, indicando que a API externa não é apenas lenta, mas também instável. Isso implica que, em um cenário de produção, a escalabilidade da solução não seria limitada pela *blockchain*, mas sim pela capacidade e confiabilidade do sistema ERP a qual ela se integra.

- **Benchmark de desempenho**

  O teste consistiu na execução de 1.000 transações de "submissão de pedido"para cada arquitetura, utilizando o executor de testes da ferramenta Insomnia, como pode ser visto na Figura 4.17. As métricas coletadas foram a latência média por requisição (tempo de ida e volta), a vazão do sistema (transações por segundo - TPS) e a taxa de sucesso das requisições. Os resultados consolidados podem ser vistos na Tabela 4.2.

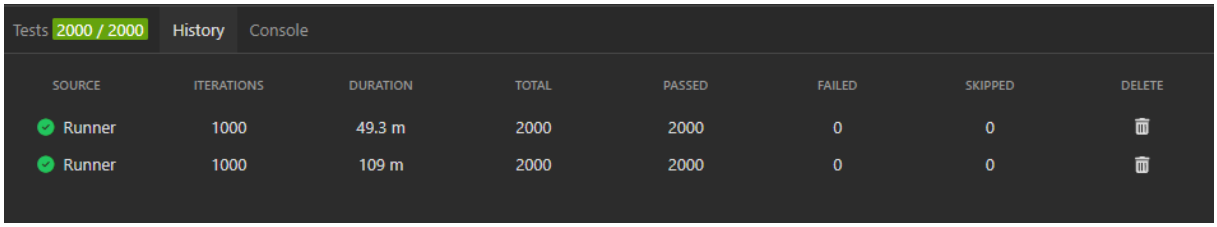


Figura 4.17 – Benchmark de desempenho - Do próprio autor (2025)

Tabela 4.2 – Tabela de comparativo de desempenho

Métrica	**Blockchain (Multichain)**	**Banco de dados tradicional (Postgres)**
Latência Média (ms)	6,540	2,958
Vazão (TPS)	0,15	0,34
Taxa de Sucesso (%)	100%	100%

Fonte: Do próprio autor (2025)

A análise dos dados evidencia que a arquitetura com banco de dados tradicional apresenta um desempenho significativamente superior, com menor latência e maior vazão. Este resultado é esperado, dado que operações em um banco de dados centralizado não incorrem no *overhead*[1] de comunicação em rede P2P(*peer-to-peer*) e do processo de consenso distribuído.

Entretando, a perda de performance observada na arquitetura blockchain é um *trade-off* deliberado e justificado pelos requisitos do problema. A utilização da MultiChain introduz garantias de imutabilidade, transparência e auditabilidade. Juntamente com a capacidade de provar criptograficamente a existência e a integridade de um documento em um determinado momento, sem depender de uma autoridade central, é o benefício que fundamentalmente justifica a adoção da arquitetura distribuída, atendendo ao objetivo principal deste trabalho.

- **Comparativo de Arquiteturas**

  Por fim, para contextualizar a arquitetura proposta em relação as diferentes abordagens discutidas na revisão bibliográfica, a comparação resumida, que pode ser vista na Tabela 4.3, destaca o posicionamento deste trabalho.

Tabela 4.3 – Comparativo de abordagens em trabalhos correlatos

Característica	**Silva (Este trabalho)**	**Manoharan (2025)**	**Sombrio et al. (2024)**
Plataforma Blockchain	MultiChain	Hyperledger Fabric	MultiChain
Execução da Lógica	Off-Chain (Servidor Flask)	On-Chain (Smart Contracts)	Off-Chain (Aplicação)
Armazenamento Off-Chain	IPFS	IPFS	Não especificado
Setor de Aplicação	Indústria Têxtil (B2B)	Cadeia de Suprimentos Química	Instituições de Ensino
Modelo de Confiança	Confiança no servidor da aplicação; verificação na blockchain	Confiança distribuída na rede de endossantes	Confiança no servidor da aplicação; verificação na blockchain
Principal Contribuição	Estudo de caso de integração com ERP	Framework de rastreabilidade	Protótipo para validação de certificados acadêmicos

Fonte: Do próprio autor (2025)

[1] *Overhead:* refere-se ao aumento do espaço necessário para armazenar um dado após sua conversão, influenciado por fatores como formatação, redundância e requisitos de compatibilidade.

5 Considerações Finais

Este capítulo final consolida as contribuições e os resultados obtidos ao longo do desenvolvimento deste trabalho. A seção de conclusão sintetiza os principais achados, validando o alcance dos objetivos propostos, enquanto a seção de trabalhos futuros delineia direções para a continuidade e expansão da pesquisa.

5.1 Conclusão

O presente trabalho teve como objetivo o desenvolvimento e a validação de um protótipo de aplicação para a autenticação de documentos na indústria têxtil, utilizando uma arquitetura que integra tecnologias de *blockchain*, *smart contract*, e APIs externas. Os resultados apresentados demonstram que a solução proposta é funcional e viável, que atende aos requisitos de segurança e integridade para o registro de transações em um ambiente B2B.

A principal contribuição desta pesquisa reside na proposição de um modelo de arquitetura para a integração de tecnologias de registro distribuído em ecossistemas corporativos previamente existentes. Por meio de testes funcionais e de desempenho, foi possível extrair conclusões significativas. O *benchmark* comparativo entre a arquitetura baseada em MultiChain e uma alternativa com banco de dados tradicional quantificou o *overhead* de performance inerente à solução distribuída. Embora a abordagem tradicional obtenha maior performance, a perda de desempenho é um *trade-off* justificado pelos ganhos em imutabilidade, transparência e auditabilidade. Adicionalmente, a análise de carga nos componentes revelou que o principal gargalo de desempenho e estabilidade do sistema não reside na *blockchain*, mas na conexão com a API do sistema ERP externo (NOMUS), um achado fundamental para melhorar a capacidade geral. Nesse contexto, a implementação da lógica de negócio de forma *off-chain* provou ser uma decisão de design possível, equilibrando os benefícios da *blockchain* e mitigando limitações existentes na capacidade de integração com sistemas externos.

O protótipo desenvolvido não apenas atendeu aos objetivos específicos de criar um sistema de autenticação de documentos, mas também serviu como um estudo de caso prático, validando um modelo de arquitetura que pode ser adaptado para outros cenários que buscam modernizar processos e aumentar a confiança entre parceiros de negócio, sem desconsiderar as restrições do mundo real.

5.2 Trabalhos Futuros

A partir dos resultados obtidos, diversas frentes de pesquisa e desenvolvimento podem ser exploradas para a evolução deste projeto:

- Implementação de arquitetura assíncrona: Com base na identificação do gargalo de desempenho na API do ERP NOMUS, uma evolução natural seria a implementação de uma arquitetura orientada a eventos para gerenciar as requisições ao ERP. Buscando desacoplando os sistemas, e assim melhorar a resiliência, a escalabilidade e o tempo de resposta da aplicação principal para o usuário.
- Migração dos *smart contracts* para *on-chain*: Se propõe a reimplementação da solução em uma plataforma de *blockchain* permissionada que suporte *smart contracts* complexos de forma nativa, como a *Hyperledger Fabric*[1]. Essa migração permitiria a execução da lógica de negócio de forma *on-chain* possibilitando maior segurança, e complexidade de desenvolvimento das regras de negócio.
- Expansão para rastreabilidade da cadeia de suprimentos: O *framework* desenvolvido pode ser expandido para cobrir outros elos da cadeia de suprimentos têxtil como a rastreabilidade da matéria-prima utilizada para o cliente específico, desde a origem (fábrica de tecidos), até o consumidor final, permitindo a criação de um atestado digital para cada lote de produtos, garantindo a qualidade dos materiais.

[1] Hyperledger Fabric:Projeto open source da *Linux Foundation* para redes *blockchain* permissionadas, com suporte a contratos inteligentes, modularidade e canais privados entre participantes.

Referências

ALI, F.; KUMAR, S.; SHARMA, P. A survey on the interplanetary file system: Concepts, applications, and challenges. *Journal of Network and Computer Applications*, Elsevier, v. 150, p. 102116, 2020.

ANOREG-CE. *Cartórios no Brasil: conheça a história do primeiro cartório do Brasil*. 2022. Disponível em: <https://www.anoregce.org.br/cartorios-no-brasil-conheca-a-historia-do-primeiro-cartorio-do-brasil/>, note = Acesso em: 05 abr. 2025.

BENET, J. *IPFS - Content Addressed, Versioned, P2P File System*. 2014. Disponível em: <https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6z7YQGd82WvH1dAsn1ph2jk6V7Ss>. Acesso em: 13 jul. 2025.

CHAUHAN, P.; VERMA, J.; JAIN, S.; RAI, R. Blockchain based framework for document authentication and management of daily business records. In: *Blockchain for 5G-Enabled IoT*. [S.l.]: Springer, 2021. p. 497–517. ISBN 978-3-030-67489-2.

CNN. *Golpistas usam deepfake de diretor financeiro e roubam US 25 milhões*. 2024. Disponível em: <https://www.cnnbrasil.com.br/economia/negocios/golpistas-usam-deepfake-de-diretor-financeiro-e-roubam-us-25-milhoes/>. Acesso em: 06 jan. 2025.

FERREIRA, E.; ALBUQUERQUE, C.; ROCHA, A.; ROCHA, L. Uso de blockchain para privacidade e segurança em internet das coisas. In: *Anais do XVII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*. Brasília: Sociedade Brasileira de Computação, 2017. p. 51. ISBN 9788576694106.

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Tese de Doutorado) — University of California, Irvine, 2000.

FREEMAN, E. T.; ROBSON, E. *Head First JavaScript Programming: A Brain-Friendly Guide*. 2. ed. Sebastopol: O'Reilly Media, 2020.

GRINBERG, M. *Flask Web Development: Developing Web Applications with Python*. 2. ed. [S.l.]: O'Reilly Media, 2018.

INC., K. *Insomnia - API Design and Testing Tool*. 2025. Disponível em: <https://insomnia.rest>. Acesso em: 17 jul. 2025.

JUNIOR, G. G.; DAHAB, R.; HENRIQUES, M. A. A. *Avaliação do mecanismo de consenso para blockchain Committeeless Proof-of-Stake*. [S.l.], 2023.

MANOHARAN, Y. Framework for document traceability in chemical supply chain using blockchain and smart contracts. Preprint, Research Square, Version 1. 2025. Disponível em: <https://doi.org/10.21203/rs.3.rs-1808889/v1>.

MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, v. 2014, n. 239, p. 2, 2014.

MONTEIRO, E. S.; MIGNONI, M. E. *Certificados digitais: conceitos e práticas*. [S.l.]: Brasport, 2007.

NAKAMOTO, S. *Bitcoin: A peer-to-peer electronic cash system*. 2008. Disponível em: <https://bitcoin.org/bitcoin.pdf>. Acesso em: 27 ago. 2025.

NOMUS. *Introdução à integração com o ERP Nomus via REST + JSON*. 2020. Disponível em: <https://ajuda.nomus.com.br/support/solutions/articles/27000045974-introduc%C3%A3o>. Acesso em: 06 abr. 2025.

PILKINGTON, M. Blockchain technology: principles and applications. In: *Research handbook on digital transformations*. [S.l.]: Edward Elgar Publishing, 2016. p. 225–253.

POLGE, J.; ROBERT, J.; TRAON, Y. L. Permissioned blockchain frameworks in the industry: A comparison. *ICT Express*, v. 7, n. 2, p. 229–233, 2021.

PORTALERP. *Nomus*. 2025. Disponível em: <https://portalerp.com/nomus>. Acesso em: 06 fev. 2025.

RELAN, K. *Building REST APIs with Flask: Create Python Web Services with MySQL*. [S.l.]: Apress, 2019.

RICHARDS, M.; FORD, N. *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol: O'Reilly Media, 2020.

SOMBRIO, M.; ANTUNES, L.; CASAGRANDE, R. A. Blockchain para autenticação de documentos em instituições de ensino. *Caderno Pedagógico*, v. 21, n. 10, p. e9811, 2024.

STANDARDS, N. I. of; TECHNOLOGY. *Advanced Encryption Standard (AES)*. Gaithersburg, MD, 2001.

TECNOLOGIA, N. *API Nomus ERP*. 2025. Disponível em: <https://documenter.getpostman.com/view/22813773/2s93JutNgM>. Acesso em: 10 mai. 2025.

TOLKIEN, J. R. R. *O Senhor dos Anéis: A Sociedade do Anel*. São Paulo: Editora Martins, 1954. Tradução de Lenita Maria Rímoli Esteves e Almiro. 4ª tiragem.

ZOU, W.; LO, D.; KOCHHAR, P. S.; LE, X.-B. D.; XIA, X.; FENG, Y.; CHEN, Z.; XU, B. Smart contract development: Challenges and opportunities. *IEEE Transactions on Software Engineering*, v. 47, n. 10, p. 2084–2106, 2021.

Anexos

ANEXO A – Inicialização

```
#!/bin/bash
set -e

CHAIN_NAME=${CHAIN_NAME:-nomus_chain}
RPC_USER=${RPC_USER:-multichainrpc}
RPC_PASSWORD=${RPC_PASSWORD:-RPC_PASSWORD}
RPC_PORT=${RPC_PORT:-RPC_PORT}
P2P_PORT=${P2P_PORT:-P2P_PORT}
DATA_DIR="/root/.multichain/${CHAIN_NAME}"

if [ ! -d "$DATA_DIR" ]; then
  multichain-util create ${CHAIN_NAME}
  sed -i "s/^rpcuser\s*=\s*.*$/rpcuser=${RPC_USER}/"  ${DATA_DIR}/multichain.conf
  sed -i "s/^rpcpassword\s*=\s*.*$/rpcpassword=${RPC_PASSWORD}/" ${DATA_DIR}/multichain.conf
  sed -i "s/^default-rpc-port\s*=\s*.*$/default-rpc-port = ${RPC_PORT}/" ${DATA_DIR}/params.dat
  sed -i "s/^default-network-port\s*=\s*.*$/default-network-port = ${P2P_PORT}/" ${DATA_DIR}/params.dat
  echo "rpcallowip=0.0.0.0/0" >> ${DATA_DIR}/multichain.conf
fi

multichaind ${CHAIN_NAME} -daemon
sleep 5
tail -f /dev/null
```

Figura A.1 – Inicialização de nó principal – do próprio autor (2025)

```
#!/bin/bash
set -e

CHAIN_NAME=${CHAIN_NAME:-nomus_chain}
SEED_HOST=${SEED_HOST}
SEED_PORT=${SEED_PORT}
NODE_RPC_PORT=${NODE_RPC_PORT}
NODE_P2P_PORT=${NODE_P2P_PORT}
RPC_USER=${RPC_USER:-multichainrpc}
RPC_PASSWORD=${RPC_PASSWORD:-password}
DATADIR="/root/.multichain"

mkdir -p ${DATADIR}/${CHAIN_NAME}
cat > ${DATADIR}/${CHAIN_NAME}/multichain.conf <<EOL
rpcuser=${RPC_USER}
rpcpassword=${RPC_PASSWORD}
rpcport=${NODE_RPC_PORT}
rpcallowip=0.0.0.0/0
EOL

INIT_OUTPUT=$(multichaind ${CHAIN_NAME}@${SEED_HOST}:${SEED_PORT} \
  -datadir=${DATADIR} \
  -rpcport=${NODE_RPC_PORT} \
  -port=${NODE_P2P_PORT} 2>&1 || true)
GRANT_ADDR=$(echo "$INIT_OUTPUT" | grep -oP "grant\s+\K[^\s]+" | head -n 1)

for i in {1..10}; do
    if multichain-cli ${CHAIN_NAME} \
        -rpcconnect=${SEED_HOST} \
        -rpcport=6466 \
        -rpcuser=${RPC_USER} \
        -rpcpassword=${RPC_PASSWORD} \
        grant ${GRANT_ADDR} connect,send,receive; then
        break
    sleep 5
done

multichaind ${CHAIN_NAME}@${SEED_HOST}:${SEED_PORT} \
  -datadir=${DATADIR} \
  -rpcport=${NODE_RPC_PORT} \
  -port=${NODE_P2P_PORT} \
  -daemon

sleep 5
until nc -z 127.0.0.1 ${NODE_RPC_PORT}; do
  sleep 2
done
tail -f /dev/null
```

Figura A.2 – Inicialização de nó secundário – do próprio autor (2025)

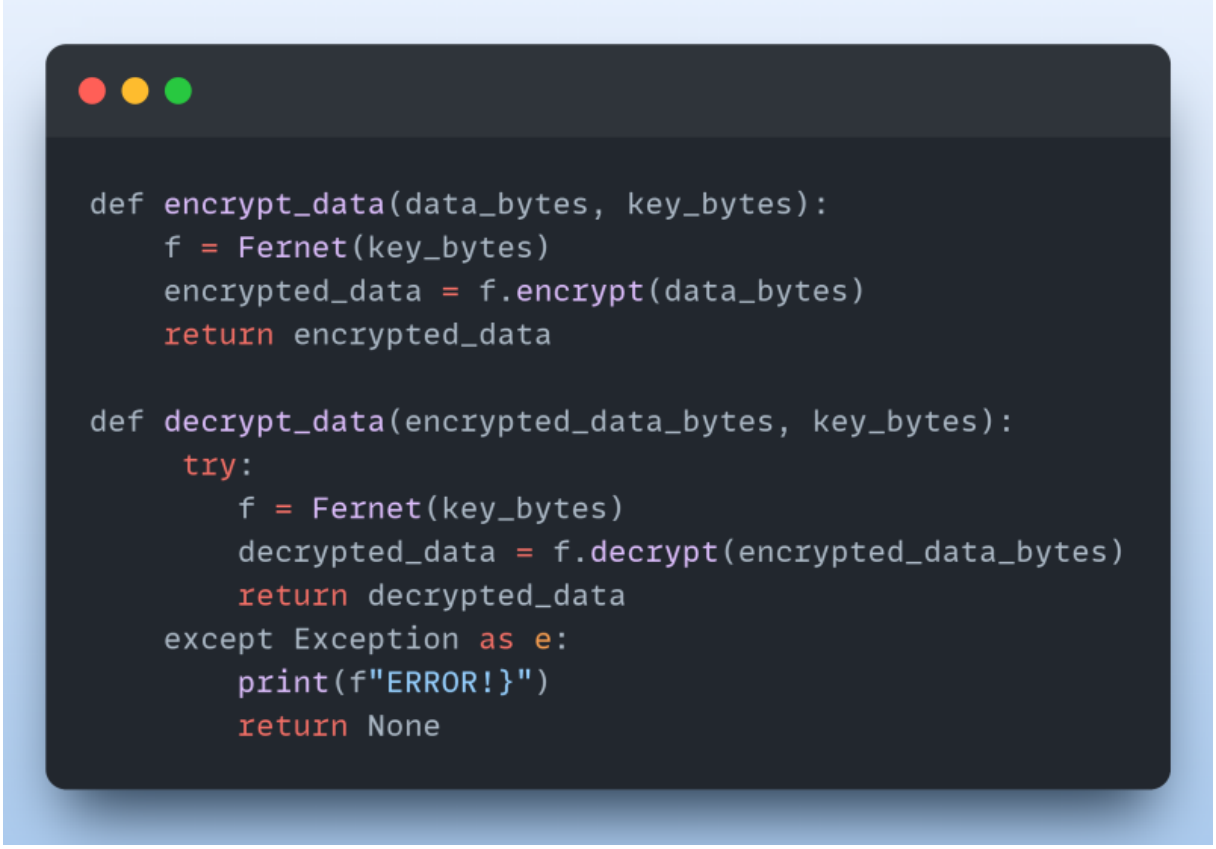
```
STREAMS_TO_CREATE = [
    'config_stream', 'inventory_stream', 'financial_stream',
    'orders_stream', 'deliveries_stream', 'notes_stream'}

def initialize_blockchain_structure():
    all_streams_ok = True
    for stream in STREAMS_TO_CREATE:
        if not blockchain_utils.create_and_subscribe_stream_if_not_exists(stream):
            All_streams_ok = False

    if not all_streams_ok:
        sys.exit(1)

def create_and_subscribe_stream_if_not_exists(stream_name):
    try:
        existing_streams = _make_rpc_request('liststreams', [stream_name, True])
        if existing_streams and any(s.get('name') == stream_name for s in existing_streams):
            return True

        create_txid = _make_rpc_request('create', ['stream', stream_name, True])
        if not create_txid:
            return False
        _make_rpc_request('subscribe', [stream_name])
        return True
    except Exception as e:
        print("Error!")
        return False
```

Figura A.3 – Inicialização de *Streams* – do próprio autor (2025)

ANEXO B – Servidor de Integração

```python
@bp.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        login = request.form.get('login')
        representante = request.form.get('representante')
        senha = request.form.get('senha')
        client_ip = request.remote_addr
        users = load_users()
        senha_hash_digitada = hashlib.sha256(senha.encode('utf-8')).hexdigest()
        user_data = next((user for user in users if user['login'] == login
        and user['representante'] == representante), None)
        if not user_data or user_data['senha_hash'] != senha_hash_digitada:
            flash('Login, identificador ou senha inválidos.', 'danger')
            return redirect(url_for('auth.login'))

        details = {}
        try:
            payload = {'client_id': user_data.get('nomus_client_id')}

            if user_data['role'] == 'cliente':
                payload['rep_id'] = user_data.get('nomus_rep_id')

            api_response = requests.post("http://localhost:5000/api/person-details",
            json=payload, timeout=10)
            api_response.raise_for_status()
            details = api_response.json()

            if user_data['role'] != 'cliente':
                details['rep_name'] = user_data.get('representante')

        except requests.exceptions.RequestException as e:
              flash("Aviso: Conexão inválida com API.", "warning")

        session.clear()
        session['user_id'] = user_data['login']
        session['user_role'] = user_data['role']
        session['user_ip'] = client_ip
        session['cliente_nome'] = details.get('client_name')
        session['cliente_cnpj'] = details.get('client_cnpj')
        session['representative_name'] = details.get('rep_name', 'N/A')
        session['senha_contract_hash'] = user_data.get('senha_contract_hash')
        session['login_time'] = datetime.datetime.now().strftime('%d/%m/%Y %H:%M:%S')
        flash('Login realizado com sucesso!', 'success')
        return redirect(url_for('request.home'))

    return render_template('login.html')
```

Figura B.1 – Rota /'login' – do próprio autor (2025)



```python
def encrypt_data(data_bytes, key_bytes):
    f = Fernet(key_bytes)
    encrypted_data = f.encrypt(data_bytes)
    return encrypted_data

def decrypt_data(encrypted_data_bytes, key_bytes):
     try:
        f = Fernet(key_bytes)
        decrypted_data = f.decrypt(encrypted_data_bytes)
        return decrypted_data
    except Exception as e:
        print(f"ERROR!}")
        return None
```

Figura B.2 – Encriptação e Decriptação de dados – do próprio autor (2025)

```python
def get_ipfs_client():
    IPFS_API_HOST = os.getenv('IPFS_API_HOST', 'ipfs')
    port = os.getenv('IPFS_API_PORT', '5001')
    try:
        client = ipfshttpclient.connect(f"/dns/{IPFS_API_HOST}/tcp/{IPFS_API_PORT}/http")
        return client
    except Exception as e:
        print(f"ERROR! {e}")
        return None

def get_from_ipfs(ipfs_hash):
    client = get_ipfs_client()
    if not client:
        return None
    try:
       data_bytes = client.cat(ipfs_hash)
        return data_bytes
    except Exception as e:
        print(f"ERROR! {e}")
        return None
```

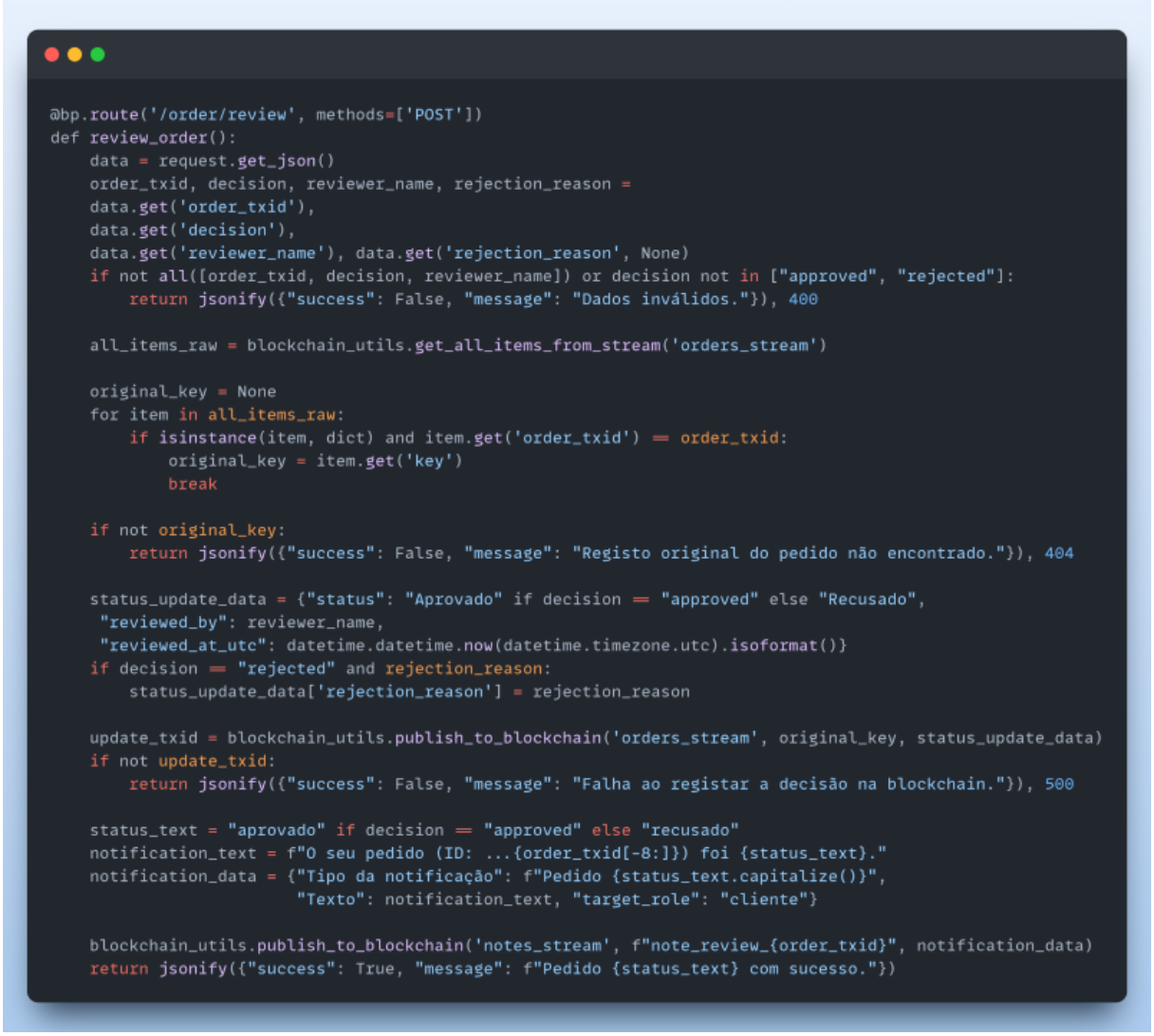
Figura B.3 – Recuperação de dados na IPFS – do próprio autor (2025)

```python
@bp.route('/order/submit', methods=['POST'])
def submit_order():
    data = request.get_json()
    if not data: return jsonify({"success": False, "message": "Dados do pedido não recebidos."}), 400
    order_items, signature_image_b64, client_info = data.get('order_items'), data.get('signature_image'), data.get('client_info')
    if not all([order_items, signature_image_b64, client_info]):
        return jsonify({"success": False, "message": "Dados do pedido incompletos."}), 400
    try:
        for group in order_items:
            for item in group.get('items', []):
                inventory_key = get_inventory_key(item.get('codigo'))
                if not inventory_key: continue
                current_inventory = blockchain_utils.get_last_item_from_stream_key('inventory_stream', inventory_key)
                if not current_inventory: continue
                new_consumed_stock = int(current_inventory.get('consumed_stock', 0)) + int(item.get('quantity', 0))
                updated_inventory_data = { **current_inventory, "consumed_stock": new_consumed_stock }
                blockchain_utils.publish_to_blockchain('inventory_stream', inventory_key, updated_inventory_data)

        signature_image_bytes = base64.b64decode(signature_image_b64.split(',')[1])
        pdf_bytes = generate_order_pdf(client_info, order_items, signature_image_bytes)
        decryption_key = os.getenv('CONTRACT_DECRYPTION_KEY').encode('utf-8')
        encrypted_pdf_bytes = ipfs_utils.encrypt_data(pdf_bytes, decryption_key)
        ipfs_hash = ipfs_utils.add_to_ipfs(encrypted_pdf_bytes)
        if not ipfs_hash: raise ConnectionError("Falha ao enviar o PDF do pedido para o IPFS.")

        order_timestamp = datetime.datetime.now(datetime.timezone.utc).isoformat()
        simplified_items = [{
            "codigo": item['codigo'],
            "modelo": item['modelo'],
            "tamanho": item['size'],
            "quantidade": item['quantity']} for group in order_items for item in group['items']]
        order_json_data = {
            "cliente": client_info.get('name'),
            "cnpj": client_info.get('cnpj'),
            "representante": client_info.get('rep_name'),
            "data_hora_utc": order_timestamp,
            "ip_origem": client_info.get('ip'),
            "produtos_solicitados": simplified_items,
            "hash_pedido_ipfs": ipfs_hash,
            "status": "Aguardando avaliação"
        }

        order_key = f"order_{client_info.get('id', 'unknown')}_{order_timestamp}"
        txid = blockchain_utils.publish_to_blockchain('orders_stream', order_key, order_json_data)
        if not txid: raise ConnectionError("Falha ao publicar o pedido na blockchain.")

        order_json_data["order_txid"] = txid
        blockchain_utils.publish_to_blockchain('orders_stream', order_key, order_json_data)

        return send_file(BytesIO(pdf_bytes), mimetype='application/pdf', as_attachment=True, download_name=f"pedido.pdf")
    except Exception as e:
        return jsonify({"success": False, "message": str(e)}), 500
```

Figura B.4 – Submissão de pedido – do próprio autor (2025)

```python
@bp.route('/delivery/submit', methods=['POST'])
def submit_delivery_proof():
    delivery_key = request.form.get('delivery_key')
    proof_file = request.files.get('proof_file')
    signature_image_b64 = request.form.get('signature_image')

    if not all([delivery_key, proof_file]):
        return jsonify({"success": False, "message": "Dados incompletos para a submissao da prova de entrega."}), 400

    try:
       signature_image_bytes = base64.b64decode(signature_image_b64.split(',')[1])
        pdf_bytes = generate_order_pdf(client_info, order_items, signature_image_bytes)
        decryption_key = os.getenv('CONTRACT_DECRYPTION_KEY').encode('utf-8')
        encrypted_pdf_bytes = ipfs_utils.encrypt_data(pdf_bytes, decryption_key)
        ipfs_hash = ipfs_utils.add_to_ipfs(encrypted_pdf_bytes)
        if not ipfs_hash: raise ConnectionError("Falha ao enviar o PDF do pedido para o IPFS.")

        update_data = {
            "status": "Aguardando aprovacao",
            "confirmed_at_utc": datetime.datetime.now(datetime.timezone.utc).isoformat(),
            "ipfs_hash_encrypted": ipfs_hash,
        }

        txid = blockchain_utils.publish_to_blockchain('deliveries_stream', delivery_key, update_data)

        if not txid:
           raise Exception("ERROR!")
        return jsonify({"success": True, "message": "Prova de entrega submetida com sucesso!", "txid": txid}), 200
        if not delivery_key or delivery_key.lower() == "null":
            return jsonify({"success": False, "message": "Chave de entrega inválida."}), 400

    except Exception as e:
        print(f"ERROR!: {e}")
        return jsonify({"success": False, "message": str(e)}), 500
```

Figura B.5 – Submissão de Entrega – do próprio autor (2025)



```python
@bp.route('/order/review', methods=['POST'])
def review_order():
    data = request.get_json()
    order_txid, decision, reviewer_name, rejection_reason =
    data.get('order_txid'),
    data.get('decision'),
    data.get('reviewer_name'), data.get('rejection_reason', None)
    if not all([order_txid, decision, reviewer_name]) or decision not in ["approved", "rejected"]:
        return jsonify({"success": False, "message": "Dados inválidos."}), 400

    all_items_raw = blockchain_utils.get_all_items_from_stream('orders_stream')

    original_key = None
    for item in all_items_raw:
        if isinstance(item, dict) and item.get('order_txid') == order_txid:
            original_key = item.get('key')
            break

    if not original_key:
        return jsonify({"success": False, "message": "Registo original do pedido não encontrado."}), 404

    status_update_data = {"status": "Aprovado" if decision == "approved" else "Recusado",
     "reviewed_by": reviewer_name,
     "reviewed_at_utc": datetime.datetime.now(datetime.timezone.utc).isoformat()}
    if decision == "rejected" and rejection_reason:
        status_update_data['rejection_reason'] = rejection_reason

    update_txid = blockchain_utils.publish_to_blockchain('orders_stream', original_key, status_update_data)
    if not update_txid:
        return jsonify({"success": False, "message": "Falha ao registar a decisão na blockchain."}), 500

    status_text = "aprovado" if decision == "approved" else "recusado"
    notification_text = f"O seu pedido (ID: ...{order_txid[-8:]}) foi {status_text}."
    notification_data = {"Tipo da notificação": f"Pedido {status_text.capitalize()}",
                         "Texto": notification_text, "target_role": "cliente"}

    blockchain_utils.publish_to_blockchain('notes_stream', f"note_review_{order_txid}", notification_data)
    return jsonify({"success": True, "message": f"Pedido {status_text} com sucesso."})
```

Figura B.6 – Revisão de Pedido – do próprio autor (2025)

```python
@bp.route('/delivery/approve', methods=['POST'])
def approve_delivery():
    data = request.get_json()
    delivery_key = data.get('delivery_key')
    reviewer_name = data.get('reviewer_name')

    if not all([delivery_key, reviewer_name]):
        print(f"ERROR!: {data}")
        return jsonify({"success": False, "message": "Dados incompletos para aprovacao."}), 400

    try:
        update_data = {
            "status": "Confirmado",
            "approved_by": reviewer_name,
            "approved_at_utc": datetime.datetime.now(datetime.timezone.utc).isoformat()
        }
        txid = blockchain_utils.publish_to_blockchain('deliveries_stream', delivery_key, update_data)

        if not txid:
            raise Exception("ERROR!")
        return jsonify({"success": True, "message": "Entrega confirmada com sucesso!"}), 200

    except Exception as e:
        print(f"ERROR!: {e}")
        return jsonify({"success": False, "message": str(e)}), 500
```

Figura B.7 – Aprovação de Entrega – do próprio autor (2025)

ANEXO C – Servidor de Requisição

```python
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'user_id' not in session:
            flash('Você precisa de estar logado para aceder a esta página.', 'warning')
            return redirect(url_for('auth.login'))
        return f(*args, **kwargs)
    return decorated_function

def role_required(allowed_roles):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            if session.get('user_role') not in allowed_roles:
                abort(403, "Você não tem permissão para aceder a esta página.")
            return f(*args, **kwargs)
        return decorated_function
    return decorator
```

Figura C.1 – Token de autenticação – do próprio autor (2025)

```
<script>
submitBtn.addEventListener('click', () => {

        const clientInfo = {
            id: "{{ session.get('user_id', 'unknown') }}",
            name: "{{ session.get('cliente_nome', 'N/A') }}",
            cnpj: "{{ session.get('cliente_cnpj', 'N/A') }}",
            rep_name: "{{ session.get('representative_name', 'N/A') }}",
            ip: "{{ session.get('user_ip', 'N/A') }}"
        };
        const payload = {
            order_items: allSelectedItems,
            signature_image: signatureImage,
            client_info: clientInfo
        };

        try {
            const response = await fetch("{{ url_for('request.proxy_submit_order') }}", {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(payload)
            });

            if (!response.ok) {
                const errorResult = await response.json().catch();
                throw new Error(errorResult.message);
            }
            alert('Pedido registado com sucesso na blockchain!');
            window.location.href = "{{ url_for('request.home') }}";

        } catch (error) {
            alert(`Erro ao registar o pedido: ${error.message}`);
    }
</script>
```

Figura C.2 – Evento de submissão de pedido – do próprio autor (2025)

```python
@bp.route('/orders/new')
@login_required
@role_required(['cliente'])
def new_order_page():
    return render_template('new_orders.html')

@bp.route('/api-proxy/order/submit', methods=['POST'])
@login_required
@role_required(['cliente'])
def proxy_submit_order():
    api_url = "http://localhost:5000/api/order/submit"
    try:
        response = requests.post(api_url, json=request.get_json(), timeout=30, stream=True)
        response.raise_for_status()
        final_headers = {k: v for k, v in response.headers.items()
                         if k.lower() in ['content-type', 'content-disposition']}
        final_headers['Access-Control-Expose-Headers'] = 'Content-Disposition'
        return Response(response.iter_content(chunk_size=1024), headers=final_headers)
    except requests.exceptions.RequestException as e:
        error_message = f"Erro no proxy de submissão: {e}"
        try:
            error_message = e.response.json().get("message", error_message)
        except (ValueError, AttributeError): pass
        return jsonify({"success": False, "message": error_message}), 502
```

Figura C.3 – Rota proxy de submissão de pedido – do próprio autor (2025)

```python
def _make_rpc_request(method, params=[]):
    user = os.getenv('MULTICHAIN_RPC_USER')
    password = os.getenv('MULTICHAIN_RPC_PASSWORD')
    host = os.getenv('MULTICHAIN_RPC_HOST')
    port = os.getenv('MULTICHAIN_RPC_PORT')

    if not all([user, password, host, port]):
        raise EnvironmentError("Variáveis de ambiente da MultiChain não configuradas)

    url = f"http://{host}:{port}"
    auth = (user, password)
    headers = {'content-type': 'application/json'}

    payload = {
        "method": method,"params": params, "jsonrpc": "2.0","id": 0,
    }

    try:
        response = requests.post(url,
                   data=json.dumps(payload),
                   headers=headers,
                   auth=auth, timeout=20)
        response.raise_for_status()
        res_json = response.json()
        if res_json.get('error'):
            print(f"ERROR: {res_json['error']}")
            return None
        return res_json.get('result')

    except requests.exceptions.RequestException as e:
        print(f"ERROR: {e}")
        if e.response is not None:
        return None

def publish_to_blockchain(stream_name, key, data_dict):
    if not create_and_subscribe_stream_if_not_exists(stream_name):
        return None

    hex_data = json.dumps(data_dict).encode('utf-8').hex()
    return _make_rpc_request('publish', [stream_name, key, hex_data])

def get_stream_item(stream_name, key):
    items = _make_rpc_request('liststreamkeyitems', [stream_name, key, False, 1])
    if items and len(items) > 0:
        hex_data = items[-1].get('data', '')
        return json.loads(bytes.fromhex(hex_data).decode('utf-8'))
    return None
```

Figura C.4 – Métodos de submissão e obtenção de dados na blockchain – do próprio autor (2025)

```python
def _make_nomus_request(method, endpoint, data=None):
    base_url = os.getenv('NOMUS_API_URL')
    api_key = os.getenv('NOMUS_API_KEY')

    if not base_url or not api_key:
       return False, {"ERROR!"}

    headers = {'Content-Type': 'application/json', 'Authorization': f'Basic {api_key}'}
    url = f"{base_url.strip('/')}/{endpoint.strip('/')}"

    try:
        response = requests.request(method, url, headers=headers, json=data, timeout=15)
        response.raise_for_status()
        return True, response.json()
    except requests.exceptions.RequestException as e:
        print(f"ERROR: {e}")


def get_nomus_deliveries(sales_order_id):
    if not sales_order_id:
        return False, {"ERROR!"}

    endpoint = f"rest/documentosEstoque/pedido/{sales_order_id}"
    return _make_nomus_request("GET", endpoint)
```

Figura C.5 – Método obtenção de pedidos na API NOMUS – do próprio autor (2025)