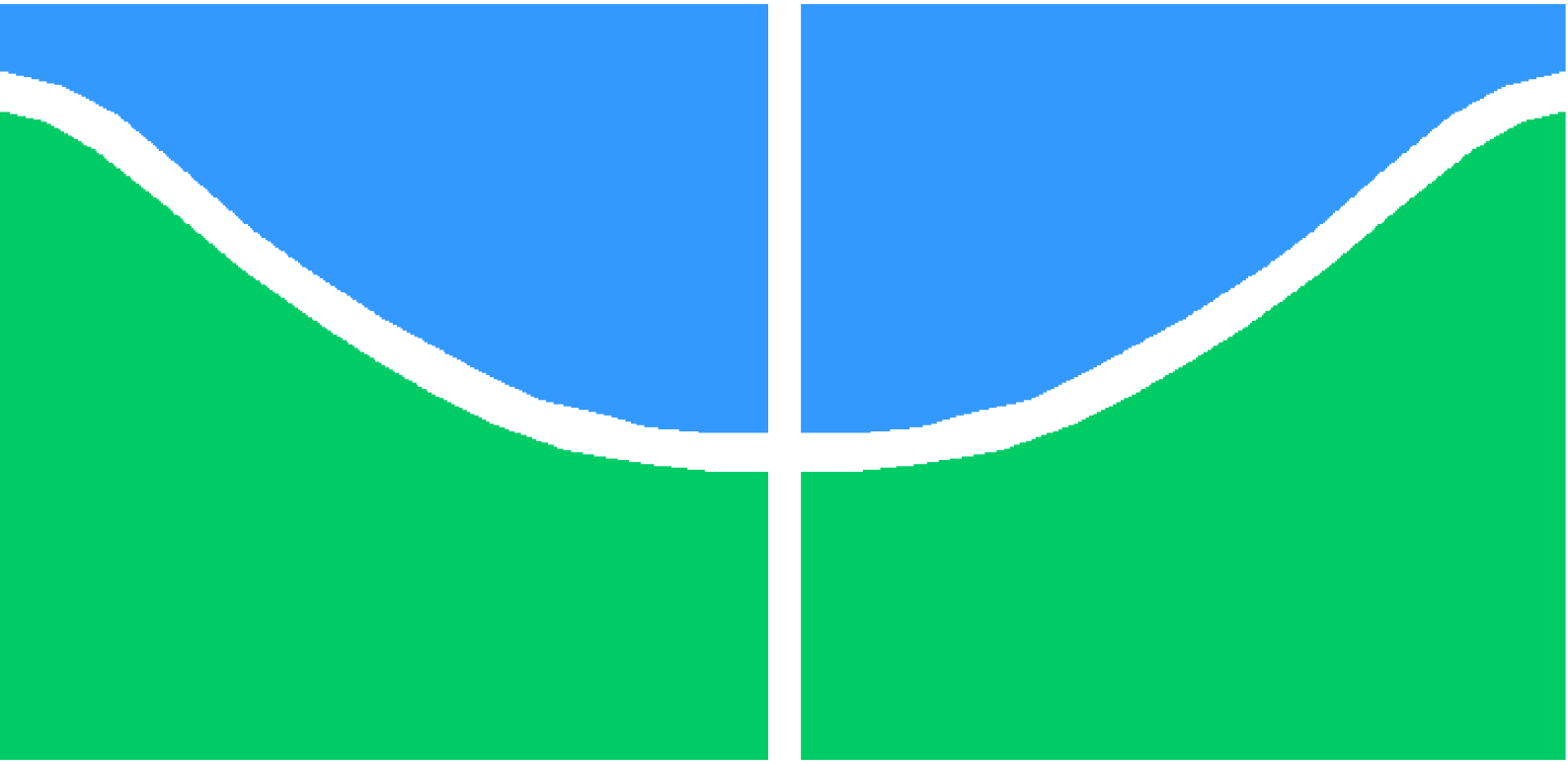


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

SmartPayment: Sistema de Automação de Contratos Inteligentes para Avaliação e Pagamento em Desenvolvimento de Software

Autor: João Pedro Elias de Moura

Orientador: Prof. Dr. Daniel Sundfeld Lima

Brasília, DF

2025

João Pedro Elias de Moura

SmartPayment: Sistema de Automação de Contratos Inteligentes para Avaliação e Pagamento em Desenvolvimento de Software

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 UnB Gama - FGA

Orientador: Prof. Dr. Daniel Sundfeld Lima

Brasília, DF

2025

João Pedro Elias de Moura

SmartPayment: Sistema de Automação de Contratos Inteligentes para Avaliação e Pagamento em Desenvolvimento de Software/ João Pedro Elias de Moura. – Brasília, DF, 2025-

79 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Daniel Sundfeld Lima

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2025.

1. Contratos inteligentes. 2. Blockchain. 3. Criptografia. 4. Criptomoeda. 5. Métricas de software. 6. Análise de pontos de função. 7. Automação de pagamentos. I. Prof. Dr. Daniel Sundfeld Lima. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. SmartPayment: Sistema de Automação de Contratos Inteligentes para Avaliação e Pagamento em Desenvolvimento de Software

CDU 02:141:005.6

João Pedro Elias de Moura

SmartPayment: Sistema de Automação de Contratos Inteligentes para Avaliação e Pagamento em Desenvolvimento de Software

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, 11 de julho de 2025:

Prof. Dr. Daniel Sundfeld Lima
Orientador

Prof. Ricardo Ajax Dias Kosloski
Convidado 1

Prof. Cristiane Soares Ramos
Convidado 2

Brasília, DF
2025

Resumo

Os contratos inteligentes, aliados à tecnologia blockchain e ao uso de moedas digitais, têm transformado a forma como transações e acordos são realizados, oferecendo segurança, transparência e automação. No contexto do desenvolvimento de software, a utilização de métricas é essencial para mensurar o esforço necessário e garantir uma compensação financeira justa entre contratantes e contratados. A Análise de Pontos de Função se destaca como uma técnica consolidada para esse fim, sendo aplicada por analistas especializados. Este trabalho apresenta o desenvolvimento do SmartPayment, um sistema web que automatiza a gestão de demandas e os pagamentos em projetos de software, integrando contratos inteligentes programados em Solidity a uma arquitetura baseada em blockchain. O sistema permite que contratantes cadastrem demandas, contratados entreguem as soluções e analistas de ponto de função atribuam pontuações com base na complexidade de cada entrega. A partir dessas pontuações, o sistema realiza automaticamente os pagamentos, utilizando criptomoedas ou moedas digitais, como ETH, USDC e a simulação da moeda digital brasileira Drex, sem a necessidade de intermediários. A aplicação foi desenvolvida utilizando Python tanto no backend quanto no frontend, com JavaScript para interações dinâmicas na interface, MongoDB para armazenamento de dados e a ferramenta Ganache como rede blockchain local de desenvolvimento. Com essa abordagem, o SmartPayment promove maior confiabilidade e eficiência nas transações financeiras, unindo métricas consolidadas da engenharia de software com tecnologias emergentes como blockchain e ativos digitais.

Palavras-chaves: Contratos inteligentes. Blockchain. Criptomoeda. Métricas de software. Análise de pontos de função. Automação de pagamentos.

Abstract

Smart contracts, combined with blockchain technology and the use of digital currencies, have transformed how transactions and agreements are executed, offering security, transparency, and automation. In the context of software development, the use of metrics is essential to measure the required effort and ensure fair financial compensation between clients and providers. Function Point Analysis stands out as a well-established technique for this purpose, applied by specialized analysts. This work presents the development of SmartPayment, a web system that automates the management of software project demands and their corresponding payments, integrating smart contracts programmed in Solidity within a blockchain-based architecture. The system allows clients to register demands, providers to deliver solutions, and function point analysts to assign scores based on the complexity of each delivery. Based on these scores, payments are processed automatically using either cryptocurrencies or digital currencies such as ETH, USDC, and a simulated version of Brazil's digital currency, Drex, without the need for intermediaries. The application was developed using Python for both backend and frontend, JavaScript for dynamic user interactions, MongoDB for data storage, and the Ganache tool as a local blockchain network for development and testing. With this approach, SmartPayment promotes greater reliability and efficiency in financial transactions, combining well-established software engineering metrics with emerging technologies such as blockchain and digital assets.

Keywords: Smart contracts. Blockchain. Cryptocurrency. Software metrics. Function point analysis. Payment automation.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

O desenvolvimento de software é uma atividade que envolve uma série de processos complexos, desde a concepção das funcionalidades até a entrega do produto final ao cliente. Nesse contexto, a gestão eficiente de contratos e a compensação financeira adequada pelos serviços prestados são desafios recorrentes para empresas e profissionais do setor. Tradicionalmente, essas atividades dependem de intermediários, como gerentes de projetos e consultores financeiros, que avaliam as entregas e definem os pagamentos. Contudo, esse modelo pode ser propenso a erros, subjetividades e atrasos, afetando a eficiência e a confiança entre contratantes e contratados (PRESSMAN; MAXIM, 2016).

A análise de software desempenha um papel importante nesse cenário, pois permite avaliar detalhadamente as funcionalidades desenvolvidas e mensurar o trabalho realizado. Entre as técnicas existentes, a análise por ponto de função se destaca por fornecer critérios objetivos e padronizados para estimar o esforço necessário à implementação de funcionalidades. Essa análise, realizada por profissionais especializados, serve como base para a definição de pagamentos, contribuindo para maior justiça e transparência entre contratante e contratado. Embora o sistema desenvolvido não automatize esse processo de análise, ele utiliza a pontuação atribuída pelo analista como referência direta para calcular os valores a serem pagos, garantindo que a compensação financeira ocorra de forma proporcional ao volume de trabalho efetivamente entregue.

A tecnologia blockchain, juntamente com os contratos inteligentes, surge como uma alternativa inovadora para automatizar e otimizar essas operações. Contratos inteligentes são programas que executam automaticamente as cláusulas estabelecidas entre as partes, eliminando a necessidade de intermediários e garantindo que os acordos sejam cumpridos de forma transparente e segura (SZABO, 1996). Aplicados ao desenvolvimento de software, esses contratos podem revolucionar a forma como os pagamentos são realizados, proporcionando maior precisão e confiabilidade na execução dos acordos (WERBACH; CORNELL, 2017). Neste trabalho, essa abordagem é utilizada para viabilizar pagamentos automáticos com base na pontuação informada por um analista de ponto de função. Como fundamento dessas tecnologias, destaca-se o uso de criptografia em blockchain e em moedas digitais, o que reforça a segurança das transações e justifica a inclusão de um capítulo introdutório sobre esse tema.

1.1 Tema

Este trabalho apresenta o desenvolvimento do SmartPayment, um sistema web que gerencia o ciclo completo de demandas e pagamentos em projetos de software, envolvendo

contratantes, contratados e analistas de ponto de função. A plataforma permite que os contratados registrem as entregas, que são posteriormente avaliadas por um analista, responsável por calcular a pontuação de pontos de função de forma externa ao sistema. Essa pontuação é então inserida na aplicação, que a utiliza como base para determinar automaticamente os pagamentos por meio de contratos inteligentes, eliminando a necessidade de intervenção humana nas transações financeiras.

O SmartPayment também integra o uso de moedas digitais, como o Drex, não apenas automatizando os processos de pagamento, mas também introduzindo novas possibilidades para transações financeiras no setor de tecnologia, contribuindo para um ecossistema mais eficiente e inovador (BANCO. . . , 2022). Nas próximas seções, serão abordados os objetivos, as metodologias e os resultados esperados com a implementação deste sistema, que visa transformar a gestão financeira de projetos de software e fortalecer a confiança entre contratantes e contratados.

1.2 Objetivo Geral

O objetivo geral deste trabalho é desenvolver um sistema web, denominado SmartPayment, que contribua com a indústria de software ao automatizar a gestão e a compensação financeira em projetos de desenvolvimento de software, utilizando contratos inteligentes baseados na tecnologia blockchain. A solução desenvolvida busca reduzir a necessidade de intermediários, garantindo maior precisão e transparência no processamento dos pagamentos, que são executados automaticamente por contratos inteligentes programados em Solidity. O sistema proporcionará um fluxo automatizado no qual contratantes solicitam demandas, contratados entregam soluções e analistas de ponto de função avaliam as entregas, gerando uma pontuação que define o pagamento automático.

1.3 Objetivos Específicos

Os objetivos específicos do trabalho incluem:

- Desenvolver uma aplicação web que permita aos contratantes gerenciar contratos e demandas, com funcionalidades específicas para cada parte interessada.
- Implementar contratos inteligentes que realizem pagamentos automáticos com base na avaliação das demandas entregues, eliminando a necessidade de intervenção humana na execução dos contratos.
- Utilizar tecnologias como Python, MongoDB e JavaScript na construção do backend e frontend do sistema, adotando uma arquitetura clara e funcional que facilite a manutenção e a extensão da aplicação.

- Promover o uso de moedas digitais, como o Drex, a moeda digital brasileira, para realizar transações financeiras no contexto do desenvolvimento de software.
- Criar um ambiente seguro e transparente para a execução de contratos, com validação automática e imutabilidade garantida pela blockchain.

1.4 Estrutura do trabalho

O presente trabalho está estruturado em sete capítulos. No Capítulo 1, apresenta-se a contextualização sobre contratos inteligentes, blockchains e a análise de software, com ênfase na análise por ponto de função, além de uma introdução ao tema e aos objetivos do estudo. Já o Capítulo 2 trata dos fundamentos da criptografia, destacando sua relevância para a segurança da informação e sua aplicação em tecnologias. Em seguida, o Capítulo 3 aborda os conceitos fundamentais de contratos inteligentes, blockchain e criptomoedas, destacando sua relevância no contexto do desenvolvimento de software. A seguir, o Capítulo 4 discute as as principais métricas utilizadas na engenharia de software, suas classificações e aplicações. O Capítulo 5 detalha a metodologia da análise de pontos de função, explicando suas etapas e a aplicação dessa técnica. Na sequência, o Capítulo 6 apresenta o sistema SmartPayment, descrevendo seu design, funcionalidades, tecnologias utilizadas, bem como as principais interfaces implementadas na solução. Por fim, o Capítulo 7 reúne as considerações finais do trabalho, incluindo uma reflexão sobre os resultados alcançados e sugestões de aprimoramentos futuros para a aplicação desenvolvida.

2 Criptografia

2.1 Criptografia

A segurança da informação tornou-se uma preocupação central na era digital. A criptografia, enquanto ferramenta essencial nesse cenário, visa proteger dados e garantir a confidencialidade, a integridade e a autenticidade da informação em ambientes computacionais e redes de comunicação. Este capítulo apresenta os principais conceitos associados à segurança de computadores, as ameaças existentes e os serviços e mecanismos desenvolvidos para mitigá-las, com ênfase nos fundamentos da criptografia.

2.1.1 Conceitos de Segurança da Informação

De acordo com o Manual de Segurança de Computadores do NIST, segurança da informação pode ser definida como a proteção de sistemas de informação automatizados, com o objetivo de preservar a integridade, a disponibilidade e a confidencialidade dos seus recursos. Esses três elementos compõem a chamada tríade CIA (*Confidentiality*, *Integrity*, *Availability*), que representa os pilares fundamentais da segurança.

A confidencialidade refere-se à garantia de que dados sensíveis não serão acessados ou divulgados a pessoas não autorizadas. Já a integridade assegura que as informações permanecem inalteradas, exceto por ações autorizadas, evitando manipulações indevidas. A disponibilidade, por sua vez, está relacionada à garantia de que os sistemas estarão acessíveis sempre que forem necessários por usuários autorizados.

Além desses princípios, outros dois conceitos são igualmente importantes: a autenticidade, que consiste na verificação da veracidade da origem das informações e da identidade dos envolvidos, e a responsabilização, que implica na possibilidade de associar ações a um agente específico, permitindo auditorias e a aplicação de medidas corretivas.

2.1.2 Ameaças e Tipos de Ataques

As ameaças à segurança da informação são classificadas, conforme as recomendações da ITU-T (*International Telecommunication Union: Telecommunication Standardization Sector*), em conjunto com a RFC 4949, em dois tipos principais: ataques passivos e ataques ativos.

Os ataques passivos visam à obtenção de informações sem que haja qualquer modificação dos dados em trânsito. São exemplos desse tipo o vazamento de conteúdo de

mensagens e a análise de tráfego. Esses ataques são difíceis de detectar, pois não alteram os fluxos de dados, mas podem ser mitigados com técnicas de encriptação.

Por outro lado, os ataques ativos envolvem alterações nos dados ou interferências no funcionamento normal dos sistemas. Eles podem ocorrer por meio de disfarce, repasse de mensagens, modificação de conteúdos e negação de serviço. Nesse caso, a detecção e a recuperação dos sistemas são medidas fundamentais, uma vez que a prevenção total é extremamente difícil.

2.1.3 Serviços de Segurança

Para enfrentar os desafios impostos pelas ameaças descritas, foram desenvolvidos diversos serviços e mecanismos de segurança. Os serviços de segurança, segundo a recomendação X.800 da ITU-T, são funcionalidades que visam proteger sistemas e transferências de dados. Os principais serviços são:

- Autenticação: garante que a comunicação seja realizada entre entidades legítimas.
- Controle de acesso: restringe o uso de recursos apenas a usuários autorizados.
- Confidencialidade: assegura que os dados não sejam divulgados indevidamente.
- Integridade: protege os dados contra modificações não autorizadas.
- Irretratabilidade: impede que remetentes ou destinatários neguem sua participação em uma comunicação.
- Disponibilidade: assegura que os recursos do sistema estejam acessíveis quando requisitados.

Esses serviços são implementados por meio de mecanismos de segurança, que podem atuar em diferentes camadas dos protocolos de rede. Dentre os mecanismos mais relevantes estão a codificação, que transforma os dados em formatos ininteligíveis; a assinatura digital, que garante a autenticidade e integridade das mensagens; a troca de autenticação, que permite verificar identidades; e o controle de acesso, responsável por impor restrições de uso. Outros mecanismos, como preenchimento de tráfego e controle de roteamento, também contribuem para mitigar riscos como a análise de tráfego e ataques de negação de serviço.

Há ainda mecanismos difusos, como trilhas de auditoria, detecção de eventos e recuperação de segurança, que complementam os anteriores ao fornecer monitoramento e resposta a incidentes.

2.1.4 Modelos para Segurança de Rede

A aplicação prática dos conceitos de segurança pode ser ilustrada por dois modelos fundamentais. O primeiro é o modelo de transmissão segura, em que a informação é protegida por meio de uma transformação criptográfica e do uso de uma informação secreta compartilhada entre as partes. Essa transformação pode incluir a encriptação da mensagem ou a adição de códigos para verificação de integridade. Em muitos casos, é necessária a presença de um terceiro confiável, responsável pela distribuição de chaves ou pela arbitragem de disputas.

O segundo modelo aborda o controle de acesso a sistemas computacionais. Ele busca proteger os recursos internos contra acessos não autorizados, utilizando métodos como autenticação por senha, filtragem de tráfego e monitoramento de atividades. Esse modelo é essencial para lidar com ameaças como vírus, worms e ataques internos maliciosos.

Esses conceitos estabelecem a base teórica necessária para compreender como sistemas de informação podem ser protegidos contra ameaças cada vez mais sofisticadas. A criptografia, nesse contexto, desempenha um papel central ao possibilitar a implementação prática de diversos serviços de segurança. A partir do próximo capítulo, será explorado com mais profundidade o funcionamento da criptografia simétrica, abordando seus principais algoritmos, características e aplicações na proteção de dados.

2.2 Criptografia Simétrica

A criptografia é o processo de transformar informações legíveis, conhecidas como texto simples, em um formato ininteligível, denominado texto cifrado. Essa transformação tem como principal objetivo proteger dados contra acessos não autorizados, assegurando sua confidencialidade mesmo quando transitam por ambientes potencialmente inseguros, como redes públicas de comunicação. A técnica permite que apenas as partes autorizadas possam reverter esse processo, mediante o conhecimento de determinadas informações secretas.

Entre os métodos criptográficos existentes, destaca-se a criptografia simétrica, também chamada de criptografia de chave secreta. Nesse modelo, uma única chave é utilizada tanto para a cifragem quanto para a decifragem das informações. Essa chave deve ser previamente compartilhada entre as partes envolvidas na comunicação e mantida sob sigilo absoluto, visto que qualquer indivíduo que venha a obtê-la poderá ter acesso pleno aos dados protegidos.

O funcionamento da criptografia simétrica inicia-se com a geração de uma chave secreta, que será utilizada por um algoritmo criptográfico para transformar o texto simples

em texto cifrado. Após essa conversão, o conteúdo cifrado é transmitido ao destinatário, que emprega a mesma chave para realizar a operação inversa e recuperar os dados originais. A segurança desse método reside na confidencialidade da chave; se essa for comprometida, todo o sistema de proteção é invalidado.

Esse modelo apresenta como principal vantagem a sua eficiência computacional. Por utilizar algoritmos menos complexos do que os métodos de chave pública, a criptografia simétrica permite o processamento rápido de grandes volumes de dados, sendo ideal para aplicações que exigem desempenho elevado. No entanto, um de seus principais desafios é o gerenciamento seguro da chave, especialmente no momento de sua distribuição entre os participantes da comunicação.

2.2.1 Algoritmos Simétricos e Cenários de Uso

A criptografia simétrica pode ser implementada por meio de dois tipos principais de algoritmos: as cifras de bloco e as cifras de fluxo. Ambas utilizam uma chave secreta única, mas diferem na forma como processam os dados e na adequação a diferentes tipos de aplicações.

As cifras de bloco operam sobre conjuntos de dados com tamanho fixo, chamados de blocos. Cada bloco de texto simples é cifrado separadamente por meio de um algoritmo que aplica transformações matemáticas complexas, produzindo blocos equivalentes de texto cifrado. Um dos algoritmos mais amplamente utilizados nessa categoria é o AES (Advanced Encryption Standard), adotado como padrão pelo governo dos Estados Unidos e por diversas organizações ao redor do mundo. Essa abordagem é especialmente adequada para proteger grandes volumes de informação armazenada ou em trânsito, como arquivos, bancos de dados e transmissões em lote.

Por sua vez, as cifras de fluxo atuam sobre o texto simples de maneira contínua, processando os dados bit a bit ou byte a byte. Essa característica torna o método particularmente útil em aplicações em tempo real, como transmissões de voz e vídeo, onde a latência e a velocidade de processamento são fatores críticos. Um exemplo clássico desse tipo de cifra é o RC4, que, apesar de sua ampla adoção no passado, teve seu uso desaconselhado em sistemas modernos devido a vulnerabilidades identificadas.

A escolha entre cifras de bloco e cifras de fluxo depende diretamente das necessidades específicas da aplicação. Sistemas que demandam alta robustez e integridade dos dados tendem a adotar cifras de bloco, enquanto contextos que exigem resposta imediata e menor atraso optam pelas cifras de fluxo. Em muitas soluções modernas, essas abordagens são combinadas de forma a equilibrar segurança e desempenho.

2.2.2 Aplicações Práticas e Desafios Atuais

A aplicação da criptografia simétrica no mundo real pode ser exemplificada por uma comunicação segura entre dois usuários, como Alice e Bob. Suponha que Alice deseje enviar um documento confidencial a Bob. Para isso, ambos devem possuir uma chave secreta compartilhada. Alice utiliza essa chave para cifrar o documento, convertendo-o em texto cifrado. Em seguida, ela transmite o conteúdo cifrado a Bob, que, utilizando a mesma chave, realiza a decifragem e recupera o texto original. O sucesso dessa comunicação depende inteiramente da preservação do sigilo da chave utilizada.

Apesar da eficácia desse método, o processo de distribuição segura da chave entre os participantes representa um desafio significativo. A transmissão da chave por meios inseguros pode comprometer toda a comunicação. Para mitigar esse risco, muitas arquiteturas modernas adotam soluções híbridas, que utilizam criptografia assimétrica apenas para o compartilhamento da chave simétrica, beneficiando-se da eficiência de ambos os métodos.

Com o avanço de tecnologias como a computação quântica, novas preocupações têm surgido em relação à longevidade dos algoritmos criptográficos tradicionais. Embora a criptografia simétrica ainda seja considerada segura, estudos indicam que certos algoritmos poderão ser mais vulneráveis a ataques de força bruta realizados por computadores quânticos. Nesse cenário, a comunidade científica e a indústria têm buscado algoritmos resistentes à criptoanálise quântica, bem como práticas que reforcem o gerenciamento seguro de chaves.

Em que pese tais desafios, a criptografia simétrica continua sendo uma ferramenta indispensável nos sistemas de segurança da informação, especialmente quando integrada a outras camadas de proteção. Sua simplicidade, eficiência e aplicabilidade em diversos contextos garantem sua relevância no cenário atual e futuro da cibersegurança.

2.3 Criptografia Assimétrica

A criptografia de chave pública representa uma das maiores inovações da história da criptografia. Diferentemente da criptografia simétrica, que utiliza uma única chave secreta para encriptação e decriptação, a criptografia de chave pública é assimétrica, utilizando duas chaves distintas, porém relacionadas: uma pública e outra privada. Esse modelo permite novas formas de proteger dados e garantir a autenticidade das comunicações.

Na criptografia de chave pública, as funções criptográficas são baseadas em operações matemáticas complexas, e não apenas em substituições e permutações, como nos sistemas tradicionais. Com esse novo paradigma, surgiram avanços significativos, sobre-

tudo na resolução de dois problemas fundamentais da criptografia: a distribuição de chaves e a autenticação digital.

Cada participante de um sistema assimétrico gera um par de chaves: a chave pública é amplamente divulgada, enquanto a chave privada é mantida em segredo. Qualquer pessoa pode usar a chave pública de um destinatário para encriptar uma mensagem, que só poderá ser decriptada por meio da chave privada correspondente, garantindo assim a confidencialidade da comunicação.

Para que os conceitos envolvidos na criptografia assimétrica fiquem mais claros, a figura 1 apresenta a terminologia essencial, incluindo definições sobre as chaves, os algoritmos e a infraestrutura necessária:

Figura 1 – Terminologia relacionada à criptografia assimétrica.

Fonte: (KISSEL, 2019)

2.3.1 Estrutura e Aplicações

Um sistema de criptografia de chave pública envolve cinco componentes básicos:

- **Texto claro:** a informação original, legível.
- **Algoritmo de encriptação:** transforma o texto claro em texto cifrado com o uso da chave pública.
- **Par de chaves:** a chave pública (disponível a todos) e a chave privada (conhecida apenas pelo proprietário).
- **Texto cifrado:** a informação encriptada, ilegível sem a chave apropriada.

- **Algoritmo de decriptação:** reverte o texto cifrado ao original utilizando a chave privada correspondente.

Além da confidencialidade, os sistemas de chave pública também oferecem suporte à autenticação, sendo utilizados em mecanismos como assinaturas digitais. O detalhamento desse recurso, incluindo sua estrutura, objetivos e aplicações, será abordado em capítulo específico mais adiante.

As principais aplicações dos criptossistemas de chave pública são:

- Encriptação e decriptação de mensagens.
- Assinatura digital de documentos.
- Troca segura de chaves: em que duas partes negociam uma chave simétrica para uso posterior.

Embora robustos, os esquemas de criptografia de chave pública também estão sujeitos à criptoanálise. Assim como na criptografia simétrica, é possível realizar ataques de força bruta, o que exige o uso de chaves suficientemente grandes para tornar o ataque impraticável. Contudo, como os algoritmos de chave pública dependem de funções matemáticas reversíveis, o aumento no tamanho da chave pode impactar significativamente o desempenho. (STALLINGS, 2015) observa: “Os sistemas de chave pública dependem do uso de algum tipo de função matemática reversível. [...] O tamanho da chave precisa ser grande o suficiente para tornar o ataque de força bruta impraticável, mas pequeno para que a encriptação e a decriptação sejam viáveis.”.

Outro risco é a possibilidade de se derivar a chave privada a partir da pública, algo que ainda não foi matematicamente descartado para nenhum algoritmo, incluindo o RSA. Por fim, ataques conhecidos como mensagem provável também representam uma ameaça específica, nos quais um invasor pode testar todas as possibilidades para mensagens simples, como uma chave DES de 56 bits. Uma medida eficaz para mitigar esse tipo de ataque é a inclusão de bits aleatórios nas mensagens antes da encriptação.

2.3.2 Algoritmo RSA

O surgimento da criptografia de chave pública criou o desafio de desenvolver algoritmos que atendessem aos seus requisitos fundamentais. O artigo pioneiro de Diffie e Hellman, publicado em 1976, apresentou essa nova abordagem e impulsionou uma busca intensa por soluções viáveis (DIFFIE; HELLMAN, 1977). Diversos algoritmos foram propostos nos anos seguintes, mas muitos se mostraram falhos ou impraticáveis. Uma das primeiras respostas bem-sucedidas veio em 1977, quando Ron Rivest, Adi Shamir e Len

Adleman, no MIT, criaram o algoritmo RSA, publicado formalmente em 1978 (RIVEST; SHAMIR; ADLEMAN, 1978). Desde então, o esquema RSA se consolidou como a técnica mais amplamente aceita e implementada para encriptação de chave pública. Trata-se de uma cifra de bloco, onde o texto claro e o texto cifrado são representados como inteiros no intervalo de 0 a $n-1$, sendo n um número grande, tipicamente com 1024 bits, ou cerca de 309 dígitos decimais.

A geração das chaves RSA envolve os seguintes passos:

1. Escolha de dois números primos grandes, p e q.
2. Cálculo de $n = p \times q$ e $\varphi(n) = (p-1)(q-1)$.
3. Escolha de um número e que seja relativamente primo a $\varphi(n)$.
4. Cálculo de d, o inverso modular de e em relação a $\varphi(n)$.

A chave pública é o par (e, n), e a chave privada é (d, n). A encriptação de uma mensagem M é dada por: $C = M^e \bmod n$, e a decriptação por: $M = C^d \bmod n$.

Embora ofereça alta segurança, o RSA é sensível a certos ataques, como os de força bruta, criptoanálise matemática e ataques de temporização. Para mitigar essas vulnerabilidades, recomenda-se o uso de tamanhos de chave adequados e técnicas de preenchimento, como o OAEP.

A criptografia de chave pública transformou a segurança da informação ao permitir confidencialidade, autenticação e troca de chaves com grande flexibilidade. O RSA, apesar de sua complexidade computacional, continua sendo um dos algoritmos mais confiáveis e amplamente aplicados.

No entanto, o avanço constante do poder computacional e dos métodos de criptoanálise exige monitoramento contínuo dos tamanhos de chave e adoção de contramedidas contra novas formas de ataque. Ainda assim, os fundamentos da criptografia de chave pública permanecem sólidos e essenciais para a segurança digital moderna.

2.4 Funções de Hash

As funções de hash criptográficas representam um componente fundamental da segurança digital moderna. Elas são projetadas para aceitar uma entrada de tamanho arbitrário, como um arquivo, mensagem ou senha, e produzir uma saída de tamanho fixo, chamada de valor de hash ou resumo da mensagem (*message digest*). Esta saída serve como uma espécie de impressão digital da informação original. “Uma “boa” função de hash tem a propriedade de que os resultados da aplicação da função a um grande

conjunto de entradas produzirá saídas que são distribuídas por igual e aparentemente de modo aleatório". (STALLINGS, 2015)

A principal característica desejada em uma função de hash é que, mesmo uma pequena alteração na entrada, como mudar uma única letra em um texto, deve resultar em uma saída completamente diferente. Essa propriedade é essencial para verificar a integridade de dados, já que qualquer modificação, acidental ou maliciosa, pode ser detectada comparando os hashes.

Em termos de desempenho, as funções de hash devem ser eficientes, ou seja, rápidas de computar mesmo em grandes volumes de dados. Além disso, elas são amplamente utilizadas em sistemas de armazenamento de senhas, assinaturas digitais e em algoritmos de integridade.

2.4.1 Propriedades de uma Função de Hash

Uma função de hash H deve, primeiramente, atender a requisitos funcionais básicos para que possa ser aplicada com sucesso em diversas aplicações computacionais. As principais propriedades gerais desejáveis são:

- **Tamanho de entrada variável:** H deve ser capaz de operar sobre entradas de qualquer tamanho, processando blocos de dados de tamanho arbitrário.
- **Tamanho de saída fixo:** independentemente do tamanho da entrada, a saída de H deve possuir um tamanho constante, geralmente definido pelo algoritmo (por exemplo, 256 ou 512 bits).
- **Eficiência:** o cálculo de $H(x)$ deve ser rápido e fácil para qualquer valor x, tanto em implementações de hardware quanto de software.

No entanto, quando se trata de uso criptográfico, são necessárias propriedades adicionais que garantam a segurança contra ataques. Essas propriedades são fundamentais para a confiabilidade das funções de hash em sistemas de integridade, autenticação e assinatura digital:

- **Resistência à pré-imagem:** dado um valor de hash h, deve ser computacionalmente inviável encontrar uma entrada M tal que $H(M) = h$.
- **Resistência à segunda pré-imagem:** dado um valor de entrada M_1, deve ser difícil encontrar outro valor M_2 tal que $H(M_1) = H(M_2)$.
- **Resistência à colisão:** deve ser difícil encontrar quaisquer duas mensagens distintas M_1 e M_2 tal que $H(M_1) = H(M_2)$.

Essas propriedades criptográficas formam a base para o uso seguro das funções de hash em aplicações como verificação de integridade, armazenamento de senhas, assinaturas digitais e códigos de autenticação de mensagem (MACs). A violação de qualquer uma dessas características comprometeria severamente a segurança dos sistemas que as utilizam.

2.4.2 Aplicações e Algoritmos

As funções de hash são utilizadas em diversas áreas da segurança da informação. Em protocolos de autenticação, elas ajudam a proteger senhas sem armazená-las diretamente. Em assinaturas digitais, o resumo de uma mensagem é assinado, e não o conteúdo completo, o que melhora a eficiência.

Diversos algoritmos foram desenvolvidos ao longo do tempo. Os primeiros algoritmos amplamente utilizados foram o MD5 (*Message Digest 5*) e o SHA-1 (*Secure Hash Algorithm 1*). No entanto, vulnerabilidades foram descobertas nessas funções, especialmente em relação à resistência a colisões.

Como resposta, a família SHA-2 foi introduzida, oferecendo variantes como SHA-256 e SHA-512, com saídas de 256 e 512 bits, respectivamente. Essas versões corrigiram as falhas das gerações anteriores e, até o momento da publicação do capítulo, permanecem confiáveis. Os principais parâmetros de cada versão da família SHA podem ser observados na Tabela 1, que resume aspectos como tamanho do resumo, tamanho do bloco, e número de etapas computacionais.

Diversas funções de hash foram desenvolvidas ao longo do tempo, muitas delas como resposta a vulnerabilidades descobertas em versões anteriores. A prática comum é ajustar algoritmos já existentes, preservando suas qualidades e melhorando sua segurança diante de novos tipos de ataques.

Figura 2 – Comparação de parâmetros do SHA.

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Tamanho do resumo da mensagem	160	224	256	384	512
Tamanho da mensagem	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Tamanho do bloco	512	512	512	1024	1024
Tamanho da word	32	32	32	64	64
Número de etapas	80	64	64	80	80

Fonte: (STALLINGS, 2015)

2.4.3 Segurança

A segurança das funções de hash é avaliada com base na complexidade computacional de encontrar colisões ou pré-imagens. Por exemplo, para uma função com saída de n bits, espera-se que encontrar uma colisão exija $2^{n/2}$ operações, enquanto encontrar uma pré-imagem exija 2^n operações.

No entanto, a história da criptoanálise mostra que ataques inovadores podem surgir, explorando propriedades inesperadas nos algoritmos. Por isso, é essencial adotar funções de hash amplamente analisadas e recomendadas por órgãos de padronização, além de manter-se atualizado com novas descobertas.

As funções de hash continuam sendo ferramentas indispensáveis nos sistemas criptográficos, sustentando a integridade, autenticação e segurança de dados nas mais variadas aplicações tecnológicas.

2.5 Autenticação de Mensagens

A proteção da confidencialidade de dados, por meio da criptografia, não garante que o conteúdo da mensagem seja autêntico ou permaneça inalterado. Nesse contexto, os códigos de autenticação de mensagem, ou MACs (*Message Authentication Codes*), surgem como mecanismos indispensáveis para assegurar a autenticidade e integridade da informação.

Um MAC é um pequeno bloco de dados gerado a partir de uma função de autenticação que depende de uma mensagem e de uma chave secreta compartilhada. Ele permite que o receptor verifique tanto a origem quanto a integridade da mensagem recebida. Essa verificação é feita recomputando o MAC com a chave compartilhada e comparando-o com o valor recebido. Caso haja divergência, a mensagem é rejeitada.

Ao contrário das assinaturas digitais, os MACs não fornecem não-repúdio, pois tanto o emissor quanto o receptor possuem a chave secreta. No entanto, essa característica também os torna eficientes e simples de implementar em ambientes que exigem comunicação segura entre partes confiáveis.

2.5.1 Objetivos e Propriedades de Segurança

O uso de MACs visa atingir três objetivos principais, fundamentais para assegurar a confiabilidade das comunicações em ambientes computacionais seguros. São eles:

- **Autenticidade:** garantir que a mensagem foi enviada por quem diz ter enviado.
- **Integridade:** assegurar que a mensagem não foi alterada durante o trânsito.

- **Verificação eficiente:** possibilitar validação com baixo custo computacional.

"A segurança de qualquer função MAC baseada em uma função de hash embutida depende de alguma maneira da força criptográfica da função de hash subjacente" (STALLINGS, 2015). Para isso, os MACs devem apresentar as seguintes propriedades de segurança:

- Dado um par mensagem-MAC, deve ser computacionalmente inviável gerar uma nova mensagem válida com o mesmo MAC.
- Deve ser impossível para um atacante, sem conhecimento da chave, gerar um par válido mensagem-MAC.
- Qualquer alteração na mensagem deve resultar em um MAC completamente diferente, impedindo falsificações.

2.5.2 Métodos de Construção de MACs

Diversas abordagens foram desenvolvidas para a construção de funções MAC seguras. As três principais são:

a) MAC baseado em cifra de bloco

Essa abordagem utiliza algoritmos como o DES ou o AES em modos especiais. O exemplo clássico é o CBC-MAC, em que a mensagem é cifrada em blocos encadeados, e o último bloco cifrado é usado como MAC. O CBC-MAC é eficaz, mas requer cuidados adicionais em mensagens de tamanhos variáveis para evitar vulnerabilidades.

b) MAC baseado em função de hash: HMAC

O HMAC (Hash-based MAC) combina uma função de hash criptográfica com uma chave secreta. A mensagem é processada em duas etapas, com a chave sendo aplicada tanto no início quanto no final do processo. O HMAC é eficiente, tem base matemática sólida e é amplamente utilizado em protocolos como SSL, IPsec e TLS.

c) MAC baseado em cifra de fluxo

Embora menos comum, cifras de fluxo também podem ser utilizadas para gerar MACs. Entretanto, seu uso exige maior atenção no controle do estado interno e sincronização, o que torna essa abordagem mais limitada.

2.5.3 Uso Prático e Protocolos

MACs são parte essencial de protocolos modernos de segurança e comunicação. Estão presentes em sistemas como:

- **SSL/TLS:** para autenticação e integridade das mensagens em conexões seguras na web.
- **IPsec:** proteção de pacotes IP em redes privadas e VPNs.
- **SSH:** utiliza autenticação e encriptação independentes, aplicando MACs ao texto claro e criptografando separadamente a mensagem, garantindo confidencialidade e integridade simultaneamente.

O uso correto de MACs evita ataques de repetição, modificação de mensagens e falsificação de remetentes. Por isso, são considerados tão fundamentais quanto a própria criptografia na segurança de sistemas distribuídos.

2.6 Assinatura digital

Nos capítulos anteriores, vimos como garantir a confidencialidade dos dados por meio da criptografia e como prover integridade e autenticação utilizando MACs e funções de hash. No entanto, em muitos contextos é necessário também assegurar que o remetente de uma mensagem não possa negar sua autoria. Para atender a esse requisito, utiliza-se a assinatura digital.

A assinatura digital é a contraparte eletrônica de uma assinatura manuscrita, oferecendo meios de verificar a autenticidade, integridade e, sobretudo, o não-repúdio de uma mensagem ou documento eletrônico. Ela é criada utilizando algoritmos de criptografia de chave pública e funções de hash, combinando o que há de mais robusto nos mecanismos apresentados anteriormente.

2.6.1 Requisitos de Segurança para Assinaturas Digitais

As assinaturas digitais, enquanto mecanismos assimétricos de autenticação, oferecem garantias mais amplas que os MACs, incluindo o não-repúdio. Para serem consideradas seguras e confiáveis, devem satisfazer os seguintes aspectos fundamentais:

- **Autenticidade:** permite ao destinatário confirmar a identidade do remetente.
- **Integridade:** qualquer alteração na mensagem invalida automaticamente a assinatura.
- **Não-repúdio:** impede que o emissor negue posteriormente a autoria da mensagem assinada.

Adicionalmente, os critérios técnicos exigem:

- Cada assinatura deve ser única para o par mensagem-emissor.
- A verificação da assinatura deve ser possível por qualquer parte que possua a chave pública correspondente.
- Deve ser impraticável criar uma assinatura válida sem a chave privada do emissor.
- A chance de encontrar duas mensagens distintas que gerem a mesma assinatura deve ser desprezível.
- A assinatura deve estar logicamente associada ao conteúdo da mensagem original.

Diferente do MAC, onde remetente e destinatário compartilham uma chave secreta, a assinatura digital utiliza um par de chaves assimétricas: o remetente assina com sua chave privada e qualquer pessoa pode verificar a assinatura com a chave pública correspondente.

2.6.2 Abordagens de Implementação

O processo de assinatura digital pode ser compreendido em duas fases complementares. Na primeira, o remetente aplica uma função de hash à mensagem original, obtendo um resumo de tamanho fixo. Esse resumo é então criptografado com sua chave privada, gerando a assinatura digital.

Na fase de verificação, o destinatário realiza o mesmo cálculo hash sobre a mensagem recebida e, em paralelo, descriptografa a assinatura utilizando a chave pública do remetente. Se os dois resumos coincidirem, a integridade da mensagem é confirmada, bem como sua autoria.

Essa abordagem tem a vantagem de permitir que a mensagem seja transmitida em texto claro, já que a segurança da comunicação não depende da confidencialidade, mas da verificação da origem e do conteúdo.

2.6.3 Integração com Funções de Hash e Criptografia

A assinatura digital é aplicada não diretamente sobre o conteúdo completo da mensagem, mas sobre um resumo criptográfico gerado por uma função de hash segura, como o SHA-256. Essa prática aumenta significativamente a eficiência, uma vez que algoritmos assimétricos são computacionalmente mais custosos.

Para prover também a confidencialidade, uma abordagem comum é combinar a assinatura com criptografia: o remetente assina a mensagem com sua chave privada e, em seguida, cifra o conteúdo com a chave pública do destinatário. Dessa forma, garantem-se simultaneamente autenticidade, integridade e sigilo, como foi discutido no capítulo 2.3.

2.6.4 Aplicações Práticas

As assinaturas digitais são amplamente utilizadas em diversos contextos. Alguns exemplos de aplicações em que elas são utilizadas incluem:

- **Certificados digitais (ex.: padrão X.509)**, nos quais uma autoridade certificadora assina a chave pública de um usuário, validando sua identidade.
- **Protocolos de segurança**, como SSL/TLS, que utilizam assinaturas para autenticar servidores em conexões seguras.
- **Sistemas de e-mail seguros**, como o PGP, que combina assinatura digital e criptografia de conteúdo.
- **Documentos legais e contratos eletrônicos**, em que a assinatura digital possui validade jurídica reconhecida em legislações nacionais e internacionais.

O uso de assinaturas digitais é essencial em ambientes que exigem responsabilidade, rastreabilidade e garantia de autoria.

3 Contrato Inteligente, Blockchain e Criptomoeda

3.1 Blockchain

Para falar de contratos inteligentes, primeiro é preciso explicar uma tecnologia maior e que é a base para a garantia da segurança, armazenamento e credibilidade dos contratos, a Blockchain. Blockchain é um livro-razão compartilhado e imutável que facilita o processo de registro de transações e rastreamento de ativos em uma rede de negócios (IBM, 2024).

As tecnologias de bancos de dados tradicionais, ordinariamente dependem que suas transações sejam autenticadas por terceiros para que seja considerada válida, como os bancos em caso de transações financeiras, cartórios para reconhecimento público de contratos ou algum outro órgão público que tenha como objetivo reconhecer que algo é legítimo. A blockchain, por sua vez, utiliza da própria rede descentralizada para garantir a legitimidade das suas transações.

Na Blockchain qualquer compartilhamento de dados, como transferências bancárias, contratos, e várias outras possibilidades, devem ser validadas por um grupo de participantes selecionados aleatoriamente que atendem aos critérios exigidos pelas diretrizes da comunidade. Eles são chamados de mineradores. A autenticação das transações na blockchain são feitas após a união de várias transações de diferentes usuários em um bloco de dados e o minerador deve achar uma hash que identifica aquele bloco. Esse processo de encontrar a hash correta para o bloco envolve muito poder computacional. Cada minerador recebe um valor considerável por bloco de dados que autentica, portanto, é algo muito disputado na comunidade, o que faz o processo ficar extremamente competitivo. Após a autenticação do minerador, o bloco só é incluído na blockchain se todo o grupo aprovar aquela hash disponibilizada pelo minerador.

Com a aprovação do grupo, o bloco é adicionado à blockchain conectando-se aos blocos anteriores e posteriores, guardando em suas informações as hashes de cada vizinho, formando assim uma cadeia de blocos. A figura 3 ilustra o encadeamento dos blocos, indicando o hash do bloco anterior, conforme é feito na Blockchain.

Figura 3 – Exemplo de encadeamento dos blocos.

Fonte: (HARTIKKA, 2017)

Podemos afirmar que é uma cadeia imutável, já que qualquer dado que tente ser alterado em um bloco da cadeia, irá alterar o valor da hash daquele bloco, o que interferirá nos blocos dos seus vizinhos, pois necessitarão atualizar o valor da hash alterada, alterando também o valor da sua própria hash e assim em diante, causando um efeito cascata, gerando inconsistências na cadeia até que ela seja totalmente alterada. Caso não sejam realizadas todas essas mudanças, a transação não será aprovada pelos outros membros da rede, e assim o bloco não será alterado na blockchain. Ou seja, para realizar qualquer alteração em um dado de um bloco, seria necessário investir tamanho poder computacional que acaba sendo algo irreal de se realizar.

Figura 4 – Representação da blockchain do Bitcoin.

Fonte: (FREEPIK, 2024)

Portanto, blockchain é uma tecnologia de banco de dados totalmente transparente a quem está conectado a rede e imutável, pois ela uma união de dados armazenados em blocos e interligados em cadeia, onde qualquer alteração feita em algum dado desses blocos causa inconsistência em toda a cadeia, o que não é permitido pelos mantenedores da rede.

Nos dias atuais, existem diversas blockchains consolidadas no mercado, tanto para uso em produção, como o Bitcoin, a Ethereum, a BSC (Binance Smart Chain), quanto é possível criar blockchains para utilizar em ambientes de desenvolvimento e simulação, onde temos exemplo da Ganache, Truffle, Hyperledger Fabric, no qual são ferramentas capazes de criar e simular blockchains.

Algumas empresas multinacionais já utilizam as blockchains em seu cotidiano. Em matéria publicada na (FORBES, 2022), são listadas as top 50 companhias bilionárias que usufruem dessa tecnologia. Algumas delas são:

Tabela 1 – Empresas que utilizam a tecnologia blockchain.

Nome da Empresa	Blockchain	Descrição
Adobe	Ethereum	"Em outubro de 2021, a empresa por trás do Photoshop e do PDF lançou o Content Attribution, que permite que os criadores exportem suas imagens diretamente para certas exchanges de tokens não fungíveis (NFTs), como KnownOrigin, OpenSea, Rarible e SuperRare."
Boeing	Go Direct, Hyperledger Fabric, Hyperledger Indy	"A Boeing está em parceria com a TrustFlight do Canadá e a desenvolvedora RaceRocks para construir o chamado sistema de registro digital de aeronaves, que ajuda as companhias aéreas a ficarem atualizadas acerca da manutenção necessária."
NBA (National Basketball Association)	Flow	"Alimentado pela blockchain "Flow" da Dapper Labs, sediada em Vancouver, British Columbia, os usuários podem comprar, vender e coletar "momentos", semelhantes a cartões comerciais digitais – como uma enterrada de LeBron James que recentemente foi vendida por um recorde de U$230.023."
Samsung	Nexledger	"Desde 2020, o braço de TI do conglomerado, Samsung SDS, usa uma plataforma de empréstimos baseada em blockchain para facilitar a solicitação de empréstimos governamentais por pequenas e médias empresas."
Walmart	Hyperledger Fabric, Walmart Blockchain	"O Walmart rastreia hoje 1.500 itens no blockchain, o triplo de um ano atrás. Suas iniciativas de segurança alimentar estão se tornando mais visíveis para os compradores: um projeto piloto recente do Sam's Club na China permitiu que os compradores digitalizassem um código QR para obter informações sobre onde o produto foi cultivado e quando foi colhido."

Fonte: (FORBES, 2022)

3.2 Criptomoedas e Moedas Digitais

Como foi citado anteriormente, na blockchain são armazenadas também transações financeiras e apesar de poder realizar transações com moedas digitais fiduciárias, sua principal e mais notória aplicação é com criptomoedas.

(STELLA, 2017) define criptomoeda como "um ativo digital denominado na própria unidade de conta que é emitido e transacionado de modo descentralizado, independente de registro ou validação por parte de intermediários centrais, com validade e integridade de dados assegurada por tecnologia criptográfica e de consenso em rede". Portanto, é uma moeda que utiliza o recurso de criptografia para verificar transações e não depende de um banco e/ou governo para emiti-la ou regulamentá-la. Importante ressaltar que atualmente existem moedas virtuais criadas por alguma instituição governamental, as chamadas Central Bank Digital Currency (CBDC), entretanto, essas não são consideradas criptomoedas devido a sua subordinação a um Estado.

A criptomoeda mais famosa e que serviu como inspiração para a criação da tecnologia blockchain, é o Bitcoin. Criado por Satoshi Nakamoto, pseudônimo usado pelo desenvolvedor ou grupo de desenvolvedores, responsável pela invenção do bitcoin em 2008, o bitcoin é uma criptomoeda, de código aberto e que funciona através de uma rede chamada peer-to-peer (ponto a ponto). Na data em que este trabalho foi escrito, uma única unidade do Bitcoin está avaliada acima de U$500.000,00, de acordo com (BINANCE, 2024). Em relatório publicado recentemente, o Banco Central Europeu reconheceu o bitcoin como uma potencial reserva de valor, assim como o é o ouro. Esse caso de uso para a criptomoeda já pode estar sendo concretizado em países com cenários econômicos adversos, em especial nas economias emergentes (MALAR, 2023).

Figura 5 – Representação do Bitcoin.

Fonte: Imagem de (STARLINE, 2024) no Freepik.

No caso das CBDCs, diversos países estão desenvolvendo e testando modelos de

moedas digitais emitidas pelo governo, e o Brasil é um deles. O Banco Central do Brasil anunciou em 2023 o projeto da moeda digital brasileira chamada Drex. Esta será apenas uma versão digital do Real, já que terá a mesma cotação, garantias e regulamentações impostas a ele. A figura a seguir traz alguns atributos do Drex:

Figura 6 – Características do Drex.

Fonte: (BANCO. . . , 2022).

O projeto do Drex tinha como previsão inicial de lançamento, para uso da população, entre o final de 2024 e início de 2025, entretanto, Rogerio Melfi, membro da ABFintechs, que acompanha o Drex de perto, acredita que esse prazo será prorrogado.

Em reportagem concedida ao InfoMoney, o BC disse que ainda pretende cumprir o prazo inicialmente definido, mas admite que para isso acontecer "o projeto e os participantes do mercado precisarão ter atingido o grau de maturidade adequado", disse a autoridade monetária, por meio de nota. (INFOMONEY, 2024)

Figura 7 – Cronograma do projeto do Drex.

Fonte: (BANCO. . . , 2023).

3.3 O que são Contratos Inteligentes?

Os contratos inteligentes estão ganhando cada vez mais notoriedade e aplicações no mundo atual, tendo em vista a ampliação do uso de tecnologias que utilizam a blockchain e os benefícios oriundos dela. Conceitualmente, Nick Szabo, um renomado criptógrafo, em um artigo publicado na década de 1990, chama um contrato de "inteligente" porque "eles são muito mais funcionais do que seus ancestrais inanimados baseados em papel. Nenhum uso de inteligência artificial está implícito" (SZABO, 1996). Ainda, define contrato inteligente como "um conjunto de promessas, especificadas em formato digital, incluindo protocolos nos quais as partes cumprem essas promessas". Apesar de Szabo ter idealizado esse tipo de contrato ainda no século passado, é possível afirmar que os contratos inteligentes só ganharam forma a partir da ascensão das blockchains.

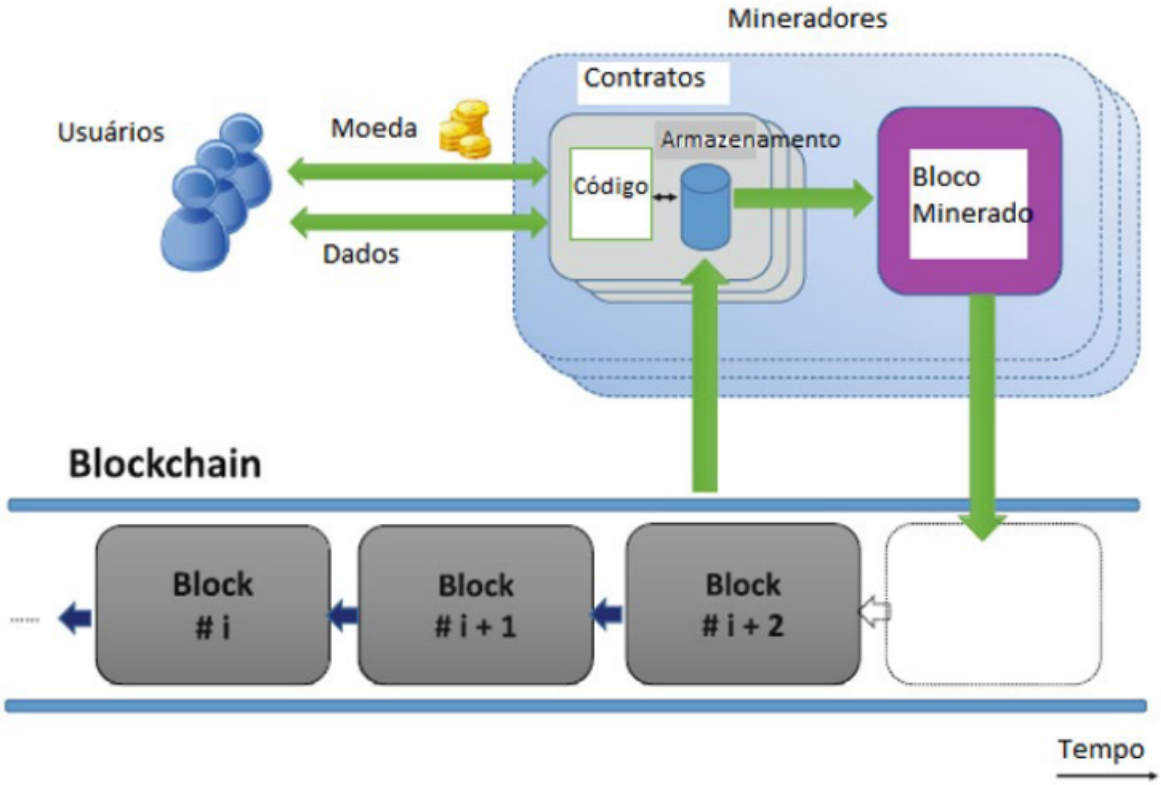
Então, atualmente, podemos simplificar o conceito definido por Szabo para, um programa executado dentro de uma rede blockchain, onde o que seriam as cláusulas de um contrato no papel, são código criptografados e comumente condicionais, que servem como gatilhos para a execução automática do contrato após serem cumpridos, sem a utilização de qualquer intervenção humana ou de inteligência artificial. Portanto, após as partes envolvidas definirem as premissas e assinarem o contrato inteligente, cabe ao sistema fazer todo o resto.

Um exemplo de contrato inteligente seria: o João tem um imóvel que deseja vender. Esse imóvel está registrado na blockchain, com todos os detalhes legais, e é associado a um token digital que representa sua propriedade. Marcos se interessa pelo imóvel e decide comprá-lo. Eles então acordam os termos da transação (preço, prazos, condições), que são

codificados em um contrato inteligente. Marcos transfere o valor acordado para o contrato inteligente. Assim que o pagamento é confirmado, o contrato inteligente automaticamente transfere a propriedade do token (e, portanto, do imóvel) do vendedor para o comprador.

Por serem feitos na blockchain, é possível garantir que os termos do contrato jamais serão alterados, possibilitando uma maior confiabilidade. O que leva a um dos muitos benefícios no uso de contratos inteligentes, que é o fim da necessidade de intermediários envolvidos no contrato apenas para verificar a validade e garantir o acordo firmado, reduzindo custos. Nenhum dos lados ou alguém mal intencionado tem o poder de realizar alterações em algum dos termos do contrato. No exemplo acima, foram eliminadas todas as taxas de corretores, cartórios e outras despesas intermediárias. A figura 8 demonstra a relação entre contrato inteligente, blockchain e criptomoedas.

Figura 8 – Relação entre contrato inteligente, blockchain e criptomoedas.



Fonte: Figura retirada de (WERBACH; CORNELL, 2017) como citado em (CAVALCANTI; NóBREGA, 2020).

4 Métricas de Software

Assim como ocorre em diversas áreas profissionais, o desenvolvimento de software também demanda métricas que permitam mensurar a contribuição dos envolvidos, a produtividade das equipes e a qualidade dos produtos entregues. Avaliar apenas se um sistema atende ou não a um determinado requisito funcional já não é suficiente em contextos que exigem estimativas precisas, controle de escopo e definição justa de esforços e compensações. Neste capítulo, são apresentadas as principais métricas utilizadas na engenharia de software, suas classificações e aplicações. Essa base conceitual permitirá, no capítulo seguinte, aprofundar o estudo de uma métrica específica amplamente utilizada em projetos de software: a Análise de Pontos de Função (APF).

4.1 Por que medir software?

A partir da Revolução Industrial, surgiu a necessidade da otimização dos processos e aumento da produção sem que houvesse um acréscimo correspondente da mão de obra. Então, foram desenvolvidos alguns métodos para a resolução desse problema, como o Taylorismo, o Fordismo e o Toyotismo. A mesma necessidade emergiu na indústria de desenvolvimento de software, após seu advento, e consequente tornou-se imprescindível a melhoria dos processos e o surgimento de métodos para aperfeiçoar esse ambiente.

No desenvolvimento de software, é comum que a equipe responsável realize um levantamento das funcionalidades que o sistema deverá possuir ou dos problemas que deverá resolver. Esses são os chamados requisitos do sistema. Em projetos de software, especialmente no que diz respeito aos requisitos não funcionais, é frequente a dificuldade de validar esses requisitos de forma objetiva. Um exemplo recorrente é o requisito de usabilidade, presente na maioria dos softwares com interação direta com o usuário. Esse requisito demanda que a interface ofereça um nível de interação agradável e satisfatória ao usuário, conforme especificado na norma (ISO/IEC, 2011).

Alguns outros requisitos para fazer a avaliação de qualidade de um software citados na norma ISO/IEC 25010, estão nas tabelas abaixo:

Tabela 2 – Características de qualidade do produto.

Características	**Sub-características**	**Definição**
Adequação Funcional	Completude Funcional	Grau em que o conjunto de funções cobre todas as tarefas especificadas e objetivos do usuário.
	Acurácia	Grau em que as funções fornecem os resultados corretos com o grau de precisão necessário.
	Funcionalidade Apropriada	Grau em que as funções facilitam a realização de tarefas e objetivos específicos.
Usabilidade	Apropriação Reconhecível	Grau em que os usuários podem reconhecer se um produto ou sistema é apropriado para suas necessidades.
	Aprendizagem	Grau em que um produto ou sistema permite ao usuário aprender como usá-lo com eficácia, eficiência em situações de emergência.
	Operabilidade	Grau em que um produto ou sistema é fácil de operar, controlar e apropriado para uso.
	Proteção Contra Erro do Usuário	Grau em que um produto ou sistema protege os usuários contra cometer erros.
	Estética de Interface com Usuário	Grau que permite uma interação agradável e satisfatória para o usuário.
	Acessibilidade	Grau em que um produto ou sistema pode ser usado por pessoas com a mais ampla gama de características e capacidades para atingir um objetivo específico em um contexto de uso específico.

Fonte: (ISO/IEC, 2011).

Tabela 3 – Características de qualidade de uso do produto.

Características	**Sub-características**	**Definição**
Efetividade		Precisão e integralidade com que os usuários alcançam seus objetivos específicos.
Satisfação	Utilidade	Grau em que um usuário está satisfeito com a percepção das conquistas sobre os objetivos pragmáticos, incluindo os resultados e as consequências do uso.
	Confiança	Grau em que um usuário ou outra parte interessada tem confiança de que um produto ou sistema se comportará como pretendido.
	Prazer	Grau em que um usuário obtém prazer em satisfazer suas necessidades pessoais.
	Conforto	Grau em que o usuário está satisfeito com o conforto físico.

Fonte: (ISO/IEC, 2011).

Mas como podemos medir esse grau? Como garantir que uma interface tenha uma interação agradável para várias pessoas completamente distintas ao mesmo tempo? Como medir a satisfação de ter suas necessidades pessoais atendidas de dois ou mais usuários?

Essas foram algumas das perguntas que impulsionaram o surgimento das métricas para um software, para que de alguma forma fosse possível chegar em algum número que consiga expressar se a característica esperada foi atingida ou não, a fim de evitar a subjetividade que pode ser gerada através das avaliações de diferentes pessoas.

Com a medição de software, é possível prever tendências positivas ou negativas, nesse segundo caso, gerando a oportunidade de corrigi-las, permite definir objetivos realistas, estimativas de esforço podem ser mais bem acuradas e, consequentemente, tudo isso contribui para um controle mais eficaz do projeto, proporcionando uma gestão mais assertiva e permitindo tomar medidas proativas para garantir o sucesso do empreendimento.

(PRESSMAN; MAXIM, 2016) descreve a engenharia de software como uma tec-

nologia em camadas que formam uma pirâmide, onde a base, o elemento que sustenta a engenharia de software, é a qualidade do software, como mostrado na figura a seguir.

Figura 9 – Camadas da engenharia de software.

Fonte: (PRESSMAN; MAXIM, 2016)

4.2 Conceitos fundamentais de métricas de software

Segundo Institute of Electrical and Electronic Engineers (IEEE), métrica é uma "medida quantitativa do grau que um sistema, componente ou processo possui um determinado atributo", ou seja, é uma forma quantitativa de aferir a qualidade de um produto. As métricas devem ser usadas como indicadores, para auxílio nas tomadas de decisões.

As demandas crescentes das aplicações de software, impulsionadas pela evolução tecnológica e pelas expectativas dos usuários, estão se tornando cada vez mais complexas, ultrapassando a capacidade das ferramentas e métodos convencionais de desenvolvimento de software. Como resultado, os projetos enfrentam desafios adicionais, como prazos mais apertados e requisitos mais complexos, o que muitas vezes leva a uma degradação na qualidade do produto final. Nesse contexto, a avaliação e mensuração criteriosas tornam-se atividades cruciais para assegurar a qualidade desejada, identificar áreas de melhoria e garantir a satisfação do cliente.

No caso específico da utilização de métricas para avaliação de um software, elas podem ser utilizadas por engenheiros de software para avaliar se há um ganho na qualidade do software entregue. Não somente isso, como citado anteriormente, também auxilia no gerenciamento de um projeto de software, a aferir a efetividade de uma equipe de desenvolvimento, a calcular o custo e tempo gastos de acordo com a complexidade do problema, além de várias outras variáveis que surgem no desenvolvimento de software. Entretanto, por mais que essas métricas tendem a buscar a objetividade, muitas vezes elas são indiretas, pois não são fundamentadas nas leis quantitativas da física, como é o caso de outras engenharias, logo, elas sempre devem estar abertas ao debate.

Deve-se sempre buscar entender o contexto que se encontra o projeto de software e definir quais métricas são apropriadas para tal. Durante o período de iniciação do uso de métricas na indústria de software, muitas delas se mostraram inadequadas para utilização

devido a sua alta complexidade e dificuldade de entendimento pela maior parte dos profissionais da área, perdendo toda praticidade que uma métrica deveria trazer. Eram gastos mais tempo entendendo as métricas do que propriamente aplicando em seus projetos e isso tornou-se ineficaz. Segundo (PRESSMAN; MAXIM, 2016), a experiência indica que uma métrica será usada somente se ela for clara e fácil de calcular. Se forem necessárias dezenas de "contagens" e cálculos complexos, é pouco provável que seja amplamente adotada.

Uma métrica eficaz deve estar alinhada com a intuição do engenheiro sobre o atributo do produto em análise. Além disso, é imperativo que a métrica gere resultados sem ambiguidades, utilizando cálculos matemáticos que evitem combinações incoerentes de unidades. Por exemplo, uma métrica que envolve o número de pessoas x linhas de código x segundo não gera uma unidade consistente, algo intuitivo para todos os engenheiros. Por fim, uma métrica deve ser independente de uma linguagem de programação e o principal, a métrica deve ter utilidade para gerar informações sobre a qualidade do produto.

4.3 Classificação de Medição de Software

Durante o período em que a indústria de software surgiu até tempos recentes, houve uma espécie de busca ao tesouro das métricas de software. Vários foram os pesquisadores que tentaram solucionar esse problema da subjetividade que as métricas de software podem gerar, buscando sempre encontrar uma métrica que levasse a uma medida que compreendesse toda a complexidade de um software, mas tudo isso foi sem sucesso.

(PRESSMAN; MAXIM, 2016) faz uma analogia para justificar essa dificuldade que os pesquisadores encontraram, onde ele diz o seguinte, "Considere uma métrica para avaliar um carro de luxo. Alguns podem destacar o projeto do chassi; outros podem destacar as características mecânicas; outros ainda podem destacar o custo ou desempenho ou o uso de combustíveis alternativos ou a facilidade de reciclagem quando o carro já estiver inutilizável. Como cada uma dessas características pode "brigar" com as outras, é difícil derivar um valor único para "atraente". O mesmo problema ocorre com o software."

Mesmo com essa complexidade, não é possível ignorar os benefícios que a medição de software proporciona. Podemos considerar que, se colocados em avaliação, as melhorias de tempo, custo e qualidade em um projeto de software orientado a métricas propiciam são instantaneamente perceptíveis.

Existem dois tipos de medições de software e que são classificados da seguinte maneira: medidas diretas e medidas indiretas. Essa classificação advém das próprias medições do mundo físico, onde as medidas diretas são aquelas as quais podemos quantificar, por exemplo, o peso de uma peça de carne é algo quantitativo, portanto é uma medida direta. Já a qualidade da peça de carne é um exemplo de medida indireta, tendo em vista

que não é algo objetivo, envolve uma relação de diferentes fatores.

Portanto, as medidas diretas no processo de software são relacionadas às características fundamentais ou básicas que a aplicação possui, como o custo, esforço gasto pela equipe de desenvolvimento, número de linhas de código, tamanho da documentação, entre outros. Em relação às medidas indiretas, algumas delas são relacionadas a qualidade, complexidade, facilidade de manutenção, pois elas são mais difíceis de serem avaliadas e ocasionando na necessidade de gerar uma relação entre diferentes coeficientes.

4.3.1 Tipos de métricas de software

Apesar de possuir diferentes categorias de métricas de software, elas devem ser usadas em conjunto para fornecer uma visão abrangente do desenvolvimento de software. Abaixo foram citados 5 tipos de métricas tradicionalmente mais conhecidas e utilizadas, são elas: métricas de processo, métricas de projeto, métricas de produto, métricas orientadas a tamanho e métricas orientadas à função.

Figura 10 – Tipos de métricas de software.

Fonte: Autor

Métricas de Processo

As métricas de processo são coletadas durante todo um projeto de software e no decorrer de longos períodos. Podemos defini-las como indicadores utilizados para melhorar o processo de software a longo prazo, como mensurar o desempenho de processos, ações e estratégias realizadas na empresa.

Métricas de Projeto

Estas são usadas, em sua maioria, pelos gerentes de projetos para adaptar o fluxo de trabalho, avaliar o estado de um projeto em andamento, rastrear os riscos em potencial, descobrir áreas problemáticas antes que se tornem "críticas", avaliar a habilidade da equipe de projeto para controlar a qualidade dos artefatos de software (PRESSMAN; MAXIM, 2016).

Métricas de Produto

As métricas de produto tem como função, ajudar a medir alguma característica de qualidade do produto como desempenho, confiabilidade, usabilidade, segurança e eficiência. Essas métricas são fundamentais para garantir que o software atenda às expectativas do usuário e aos requisitos de qualidade, além de fornecer insights valiosos para o processo de desenvolvimento e manutenção do software.

Métricas Orientadas a Tamanho

Métricas de software orientadas a tamanho são criadas pela normalização das medidas de qualidade e/ou produtividade, levando-se em consideração o tamanho do software produzido (PRESSMAN; MAXIM, 2016). O exemplo mais familiar e intuitivo dessa métrica é a contagem de linhas de código do software (LOC, do inglês *Lines of Code*).

Figura 11 – Métricas orientadas a tamanho.

Projeto	LOC	Esforço	US$(000)	Pág. doc.	Erros	Defeitos	Categoria
alfa	12.100	24	168	365	134	29	3
beta	27.200	62	440	1224	321	86	5
gama	20.200	43	314	1050	256	64	6
⋮	⋮	⋮	⋮	⋮	⋮		

Fonte: (PRESSMAN; MAXIM, 2016)

A partir dessa tabela podemos derivar algumas métricas, como:

- Produtividade = Erros / LOC
- Qualidade = Defeitos / LOC
- Custo = US$ / LOC
- Documentação = Páginas de documentação / LOC

Essa métrica é mais utilizada para obtenção de informações sobre o projeto, porém, não é muito bem aceita na comunidade de engenheiros de software para o uso nas estimativas, pois ela pode penalizar um código bem projetado e enxuto, além de ser totalmente dependente da linguagem de programação utilizada, que como citado na seção 4.2, não é uma característica de uma boa métrica.

Métricas Orientadas a Função

Esse tipo de métrica tem um princípio semelhante ao tipo anterior, porém concentra-se na utilização como valor de normalização uma medida da funcionalidade do software. Ou seja, baseia-se na avaliação do valor das funções presentes nos códigos das aplicações. O capítulo a seguir será dedicado a apresentar uma técnica desse tipo de métrica, chamada: Análise de Pontos de Função.

5 Análise de Pontos de Função

5.1 Origem

Em virtude da complexidade e da insegurança que as avaliações por linha de código traziam para os engenheiros de software na década de 1970, Allan Albrecht, um integrante da *International Business Machine* (IBM) na época, elaborou um método que pudesse avaliar a produtividade de um dos projetos de software da empresa, entretanto, que não fosse tão dependente da linguagem de programação utilizada. O método foi apresentado à comunidade e prontamente aceito e utilizado no meio, destacando-se os trabalhos de Capers Jones, um importante engenheiro na área de análise de software até os dias atuais.

Em consequência disso, a adesão de usuárias da técnica de ponto de função foi tamanha, que em 1986 fundou-se o IFPUG (do inglês, *International Function Point Users Group*), uma entidade sem fins lucrativos, composta por especialistas em análise de pontos de função e membros presentes em 30 países, cujo objetivo é auxiliar empresas a aperfeiçoarem seu processo de desenvolvimento de software utilizando técnicas de medições de software.

Nos anos seguintes a sua fundação, a entidade desenvolveu várias versões de um Manual de Práticas de Contagem. Em 1990, o IFPUG publicou a versão 3.0 do seu manual e essa é a versão mais aceita no mercado para contagem de pontos de função. Surgiram outros manuais de prática de contagem que também tiveram destaque na comunidade, como o Cosmic e Mark-II. As organizações responsáveis por eles disponibilizam gratuitamente os seus manuais para quem desejar ter acesso, uma prática não utilizada pelo IFPUG, onde apenas seus membros têm acesso gratuito aos manuais, o que sempre foi um complicador na dissipação da técnica elaborada por eles.

Outra associação que teve sua parcela significativa na consolidação da técnica de análise de ponto de função foi a NESMA (do inglês, *Netherlands Software Metrics Users Association*). Fundada com objetivos semelhantes ao IFPUG, também baseou-se nos manuais dela para elaborar seus próprios, utilizando alguns conceitos e regras semelhantes, porém com “uma extensão refinada do método apresentado pelo IFPUG” (VAZQUEZ; SIMÕES; ALBERT, 2018).

No Brasil, a técnica de análise de ponto de função começou a ser conhecida e ter relevância a partir de alguns encontros nacionais de usuários de pontos de função, no início da década de 1990. Outros fatores que auxiliaram bastante na difusão da técnica no Brasil foi a criação da *Brazilian Function Point Users Group* (BFPUG) e a partir de uma maior preocupação na utilização das métricas de software por parte das empresas e

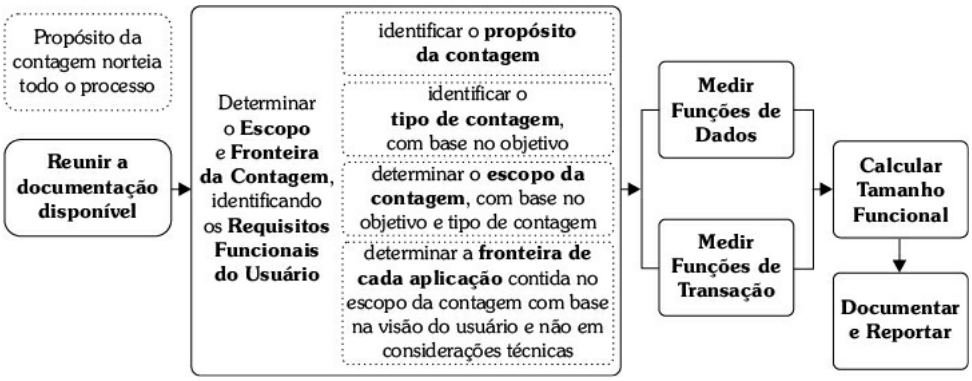
em grandes contratos públicos de desenvolvimento e manutenção de sistemas.

5.2 Etapas da Medição Funcional - IFPUG

Para realizar a medição funcional ou fazer a análise de pontos de função de um projeto, é necessário entender que deve ser levado em consideração apenas os requisitos funcionais do software, ou seja, a quantidade de pontos de função levantadas refletem somente a dimensão funcional do projeto, também conhecido como tamanho funcional do projeto. Contudo, caso queira obter estimativas de esforço, custo, prazo pode-se utilizar dos pontos de função para fazer a medição da parte funcional do software adicionado a outras variáveis que componham a parte não funcional do software.

O processo de medição funcional desenvolvido pelo IFPUG, envolve várias etapas, que serão abordadas individualmente a seguir neste capítulo, e que são representadas no diagrama abaixo:

Figura 12 – Visão geral do processo de medição funcional do IFPUG.



Fonte: (VAZQUEZ; SIMÕES; ALBERT, 2018)

5.2.1 Identificar o Propósito da Contagem

Esta etapa consiste em descobrir qual é o objetivo da análise e para qual contexto ela será utilizada. Com essa identificação, evita-se que a análise perca sua utilidade no caso de ser calculada em um contexto diferente do esperado, dado que dependendo do propósito, a resposta pode variar. Fazendo uma analogia com medições no mundo físico, caso seja requisitado o tamanho de uma casa, o resultado esperado tendo em vista que o objetivo final seja trocar o piso da casa é diferente se porventura o objetivo for pintar as paredes.

5.2.2 Reunir a Documentação Disponível

Assim como em todo o projeto de software, a documentação é uma parte vital do processo para facilitar o acesso a informações de todo o projeto. Não é diferente para o

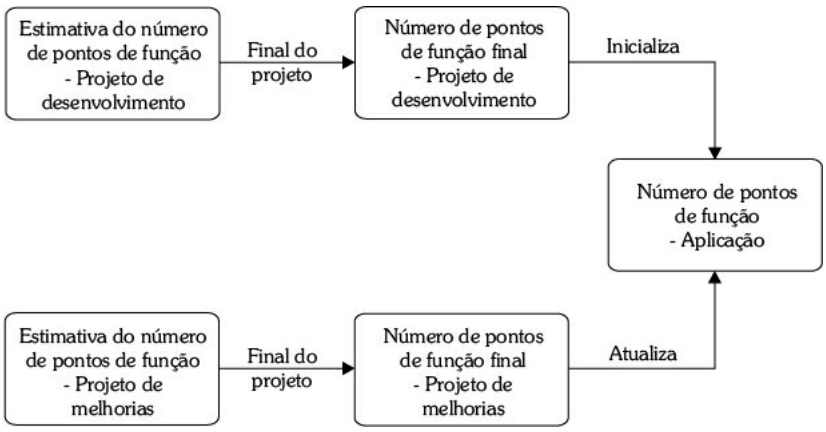
caso da análise de pontos de função. É necessário reunir a documentação para que seja identificado quais são os requisitos do software que deverão ser utilizados como base. Alguns documentos importantes que podem ser usados na medição são: diagrama de casos de uso, diagrama de classe, documento de visão. Uma documentação bem feita e organizada diminui o tempo perdido com a compreensão objetivo do produto.

5.2.3 Identificar o Tipo de Contagem

Nessa etapa os responsáveis pela medição definem qual tipo de contagem será utilizado na medição. Existem três tipos de contagem, são eles:

- **Contagem de um projeto de desenvolvimento**: O tamanho funcional de um projeto de desenvolvimento mede a funcionalidade fornecida aos usuários finais do software quando da sua primeira instalação, e abrange também as eventuais funções de conversão de dados necessárias à implantação do sistema (VAZQUEZ; SIMÕES; ALBERT, 2018).
- **Contagem de um projeto de melhoria**: O número de pontos de função de um projeto de melhoria mede as funções adicionadas, modificadas ou excluídas do sistema pelo projeto, e também as eventuais funções de conversão de dados (VAZQUEZ; SIMÕES; ALBERT, 2018).
- **Contagem de uma aplicação**: Esta contagem fornece uma medida da atual funcionalidade incluída na aplicação. Ela é iniciada ao final da contagem do número de pontos de função do projeto de desenvolvimento, sendo atualizado no término de todo projeto de melhoria que altera a funcionalidade da aplicação.

Figura 13 – Relacionamento entre os tipos de contagem.



Fonte: (VAZQUEZ; SIMÕES; ALBERT, 2018)

5.2.4 Determinar o Escopo da Contagem

O escopo é o conjunto de funções que devem ser analisadas na contagem e consequentemente os tipos de contagem são essenciais para essa definição, já que eles determinam se serão medidas funções de um projeto em desenvolvimento, de melhoria ou de uma aplicação já pronta.

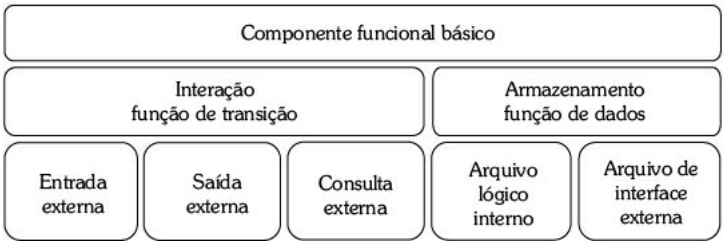
5.2.5 Determinar a Fronteira da Aplicação

Determinar a fronteira da aplicação é uma das etapas mais importantes do processo, já que caso haja uma delimitação incorreta, várias funções podem ser deixadas de fora na medição ou medidas erroneamente, o que acarretará que todo o trabalho de contagem posterior pode ser inválido. (VAZQUEZ; SIMÕES; ALBERT, 2018) define a fronteira da aplicação como "a interface conceitual que delimita o software que será medido e o mundo exterior (seus usuários)".

5.2.6 Medição das Funções

(VAZQUEZ; SIMÕES; ALBERT, 2018) define a medição por análise de ponto de função consiste em decompor o projeto ou a aplicação em elementos denominados componentes funcionais básicos (ou funções). O método conceitua abstrações, os tipos de função, nos quais os componentes funcionais básicos são classificados. Dois tipos principais de função são medidos, sendo funções de armazenamento e funções de transação. Veja na figura a seguir:

Figura 14 – Classificação dos tipos de funções.



Fonte: (VAZQUEZ; SIMÕES; ALBERT, 2018)

As funções de transação ou de processamento, são classificadas em:

- **Entrada externa**: Representa uma transação que deseja criar/atualizar dados internos do sistema a partir de informações originadas de fora da fronteira.
  Exemplo: Criar/alterar senha do usuário.
- **Saída externa**: É uma transação de envio de dados ou informações de controle oriundos do sistema, que passaram por uma lógica adicional de processamento, e são enviados para fora da fronteira.

Exemplo: Relatório com a soma do lucro mensal de uma empresa no período de 3 meses.

- **Consulta externa**: Similar a saída externa, porém sem a necessidade da lógica adicional de processamento.
  Exemplo: Relatório com o nome de todos os funcionários da empresa.

Já as funções de armazenamento, são classificadas em:

- **Arquivo Lógico Interno (ALI)**: Grupo lógico de dados, reconhecido pelo usuário, e mantidos dentro da fronteira da aplicação. Em geral encontramos os ALIs implementados fisicamente como tabelas de banco de dados atualizadas pela aplicação. Exemplo: Dados de monitoramento de entrada e saída de funcionários.
- **Arquivo de Interface Externa (AIE)**: Grupo lógico de dados, reconhecido pelo usuário, e apenas referenciados pela aplicação, ou seja, mantidos fora da fronteira. Exemplo: Dados do funcionário que não estão sendo mantidos naquele sistema.

Para deixar mais claro a diferença entre os dois tipos de funções de armazenamento, pensemos no seguinte exemplo: Um sistema de monitoramento de uma empresa necessita realizar transações que apresentem todas as entradas e saídas, nome, CPF, matrícula e cargo de um determinado funcionário. O sistema de monitoramento mantém apenas os dados de entrada e saída do funcionário e o sistema de cadastro mantém todos os outros. Portanto, para o sistema de monitoramento, o grupo de dados pessoais do funcionário é um AIE e para o sistema de cadastro, o grupo de dados pessoais do funcionário é um ALI.

5.2.7 Cálculo do Tamanho Funcional

Após a realização da identificação do propósito, a escolha do tipo de contagem e a classificação das funções, é tempo de realizar o cálculo do tamanho funcional do projeto. Para cada função de dados ou transação, há um peso em Ponto de Função (PF) a ser determinado pela complexidade da função em baixa, média ou alta. O cálculo final é realizado a partir do somatório da quantidade de Pontos de Função identificados.

Na seção seguinte será apresentado como realizar o cálculo de cada função do tipo de dados ou de transação.

5.3 Como Calcular as Funções?

O cálculo do tamanho funcional do projeto de software consiste em 3 partes. Primeiro define-se a complexidade das funções, seguido do cálculo da contribuição de cada

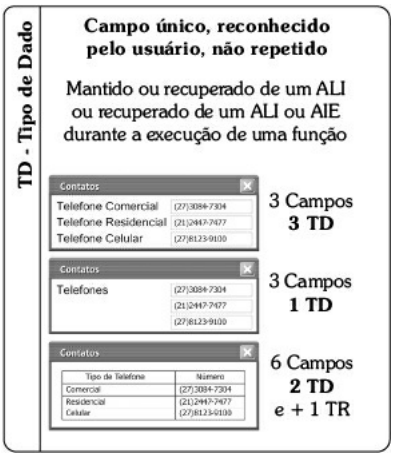
processo e por último a aplicação na fórmula disponibilizada pelo manual do IFPUG, a depender do tipo de contagem definido anteriormente.

5.3.1 Complexidade das Funções do Tipo Dado

Após a definição das funções (seção 5.2.6), é preciso calcular quanto PFs valem cada ALI e AIE. Para isso, primeiro avalia-se a complexidade funcional de cada arquivo (ALI ou AIE), utilizando de dois parâmetros:

- **Número de Tipos de Dados (TD)**: Campo único, reconhecido pelo usuário e não repetido. Não há a necessidade de identificá-los de maneira exata.

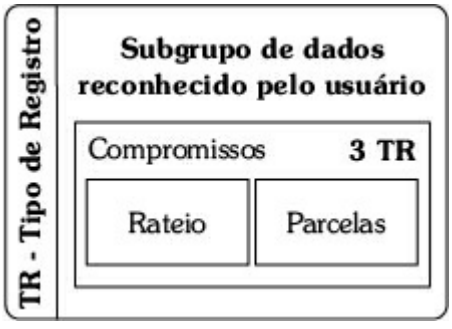
Figura 15 – Diferença entre tipo de dado e campo de um arquivo.



Fonte: (VAZQUEZ; SIMÕES; ALBERT, 2018)

- **Número de Tipos de Registro (TR)**: Grupo de elementos de dados, reconhecido pelo usuário, dentro de um ALI ou AIE. São opcionais ou obrigatórios, a depender se o usuário tem ou não a obrigatoriedade de informar esses dados na transação de criar/adicionar dados ao arquivo.

Figura 16 – Exemplo de tipos de registro.



Fonte: (VAZQUEZ; SIMÕES; ALBERT, 2018)

Após definido a quantidade de tipos de dados e registros, a complexidade deve ser medida com base nas informações da tabela abaixo:

Figura 17 – Complexidade funcional dos ALI e AIE.

TR \ TD	<20	20-50	>50
1	Baixa	Baixa	**Média**
2-5	Baixa	**Média**	Alta
>5	**Média**	Alta	Alta

Fonte: Guia de contagem de pontos de função.

5.3.2 Complexidade das Funções do Tipo Transação

Para classificar a complexidade funcional das funções de transação (EE, SE, CE), deve-se utilizar o número de Tipos de Dados (TD) como parâmetro, similar ao TD nas funções do tipo Dado, correlacionado com o número de Arquivos Referenciados (AR). Um AR é um arquivo, interno ou proveniente de uma interface externa, lido pela função do tipo transação.

Figura 18 – Complexidade funcional para Entrada Externa.

AR \ TD	<5	5-15	>15
<2	Baixa	Baixa	**Média**
2	Baixa	**Média**	Alta
>2	**Média**	Alta	Alta

Fonte: Guia de contagem de pontos de função.

Figura 19 – Complexidade funcional para Saída e Consulta Externa.

AR \ TD	<6	6-19	>19
<2*	Baixa	Baixa	**Média**
2-3	Baixa	**Média**	Alta
>3	**Média**	Alta	Alta

Fonte: Guia de contagem de pontos de função.

*Por definição, uma CE deve referenciar ao menos 1 arquivo lógico.

5.3.3 Calcular Contribuição

Definido a complexidade das funções, é possível determinar a contribuição de cada processo, isto significa, calcular quantos pontos de função as funções do tipo dado e transação correspondem. Vide a tabela 4:

Tabela 4 – Contribuição dos pontos de função das funções do tipo dado e transação.

Tipo de Função	**Baixa**	**Média**	**Alta**
Arquivo Lógico Interno (ALI)	7 PF	10 PF	15 PF
Arquivo de Interface Externa (AIE)	5 PF	7 PF	10 PF
Entrada Externa (EE)	3 PF	4 PF	6 PF
Saída Externa (SE)	4 PF	5 PF	7 PF
Consulta Externa (CE)	3 PF	4 PF	6 PF

Fonte: Guia de contagem de pontos de função.

5.3.4 Fator de Ajuste

A princípio, o fator de ajuste era uma das variáveis a serem adicionadas no cálculo do tamanho funcional de um projeto de software, contudo, atualmente não faz mais parte do processo de medição. Ainda assim, o valor do fator de ajuste permanece presente no Manual de Práticas de Contagem do IFPUG, tanto para fins de avaliação em provas de certificação como especialista em pontos de função, devido às suas regras e definições serem objeto de questões, quanto para manter a compatibilidade com contagens realizadas em versões anteriores da metodologia.

5.3.5 Cálculo do Tamanho Funcional

Por último, no uso cotidiano já existem ferramentas de apoio disponíveis aos analistas para realizar o cálculo final do projeto. Entretanto, será apresentado a seguir, para valor de conhecimento, as fórmulas descritas no manual do IFPUG para realizar o cálculo final para os três tipos de contagem: Projeto de desenvolvimento, Projeto de melhoria e Projeto de Aplicação.

Antes, é preciso definir o que é uma função de conversão, tendo em vista que essa a contagem dessa funcionalidade faz parte da fórmula do Projeto de Desenvolvimento e do Projeto de Melhoria. As funcionalidades de conversão são as funções disponíveis no momento da instalação da aplicação para converter dados ou fornecer outros requisitos de conversão especificados pelo usuário, como relatórios de verificação de conversão. Após a instalação essas funções são descartadas por não serem mais necessárias (VAZQUEZ; SIMÕES; ALBERT, 2018).

Fórmula do Projeto de Desenvolvimento

$$DFP = ADD + CFP \tag{5.1}$$

Onde:

- DFP = Contagem de pontos de função do projeto de desenvolvimento.
- ADD = Tamanho das funções a serem entregues ao usuário pelo Projeto de Desenvolvimento.
- CFP = Tamanho das funções de conversão.

Fórmula do Projeto de Melhoria

$$EFP = ADD + CHGA + CFP + DEL \tag{5.2}$$

Onde:

- EFP = Contagem de pontos de função do projeto de melhoria.
- ADD = Tamanho das funções incluídas pelo projeto de melhoria.
- CHGA = Tamanho das funções alteradas pelo projeto de melhoria.
- CFP = Tamanho das funções de conversão.
- DEL = Tamanho das funções excluídas pelo projeto de melhoria.

Fórmula do Projeto de Aplicação

$$AFP = ADD \tag{5.3}$$

Onde:

- AFP = Contagem de pontos de função do projeto de aplicação.
- ADD = Tamanho das funções entregues ao usuário.

Existe uma outra fórmula que é utilizada após o projeto de melhoria ser finalizado, já que o número de pontos de função da aplicação deve ser atualizado para refletir as modificações na aplicação. A fórmula é:

$$AFPA = (AFPB + ADD + CHGA) - (CHGB + DEL) \tag{5.4}$$

Onde:

- AFPA = Contagem de pontos de função do projeto de aplicação após a melhoria.
- AFPB = Tamanho da aplicação antes da melhoria.
- ADD = Tamanho das funções incluídas pelo projeto de melhoria.
- CHGA = Tamanho das funções alteradas pelo projeto de melhoria depois de seu término.
- CHGB = Tamanho das funções alteradas pelo projeto de melhoria antes de seu término.
- DEL = Tamanho das funções excluídas pelo projeto de melhoria.

5.4 Comparação dos Métodos de Medição de Tamanho Funcional: Nesma e IFPUG

Os métodos de Medição de Tamanho Funcional (FSM) de Nesma e IFPUG são duas abordagens amplamente utilizadas e reconhecidas internacionalmente, estando em conformidade com a norma ISO/IEC 14143-1. Ambos os métodos têm como base as diretrizes originais de A.J. Albrecht e, ao longo do tempo, passaram por diversas revisões que os aproximaram, embora algumas diferenças persistam. Em 2004, a IFPUG estabilizou suas diretrizes com o lançamento do IFPUG CPM 4.2, e, desde então, as atualizações são mínimas. As diretrizes da Nesma também se mantêm estáveis desde a versão 1.0 de seu manual, publicada em 1989. Atualmente, as diferenças entre os métodos são poucas, com impacto relativamente pequeno na contagem final de pontos de função para sistemas e projetos (NESMA, 2019).

5.4.1 Versões Atuais dos Manuais Nesma e IFPUG

As versões atuais dos manuais de contagem são o Nesma CPM 2.3 (ISO/IEC. . . , 2018), e o IFPUG CPM 4.3 (ISO/IEC. . . , 2009). Ambos os métodos utilizam terminologias similares e critérios quase idênticos para identificar processos elementares e funções de dados, diferenciando-se nos seguintes aspectos: o tratamento de Inquéritos Externos versus Saídas Externas, características gerais do sistema, consultas implícitas, tratamento de tabelas de códigos e a abordagem a mídias físicas. Essas variações impactam mais na categorização das funções do que no número total de funções contadas (NESMA, 2019).

5.4.2 Principais Diferenças entre Nesma e IFPUG

As principais diferenças incluem a definição de Inquéritos Externos e Saídas Externas, onde a IFPUG conta certas funções como Inquéritos Externos, enquanto a Nesma pode considerá-las Saídas Externas, dependendo de critérios específicos. Outras distinções incluem a exclusão das características gerais do sistema nas diretrizes da Nesma desde a versão CPM 2.3, enquanto a IFPUG mantém essas características em um apêndice. Em relação a consultas implícitas, a Nesma tende a não considerá-las como funções transacionais separadas, enquanto a IFPUG as conta se identificadas explicitamente pelo usuário. Para tabelas de códigos, a Nesma as classifica como um tipo de função, ao passo que a IFPUG não as considera parte dos requisitos funcionais do usuário (NESMA, 2019).

6 Desenvolvimento da Solução

O objetivo deste trabalho foi desenvolver um sistema para automatizar a gestão e compensação financeira em projetos de desenvolvimento de software, utilizando contratos inteligentes. A solução visa ajudar a diminuir a utilização de intermediários para a execução de contratos, aumentar a transparência nos pagamentos e permitir a definição prévia dos valores a serem pagos por entregas realizadas. Além disso, o sistema foi concebido com compatibilidade para uso com moedas digitais, como o Drex, atualmente em desenvolvimento no Brasil.

Atualmente, contratantes e contratados enfrentam dificuldades em quantificar e valorar o esforço de desenvolvimento de maneira justa e objetiva. A aplicação desenvolvida oferece um fluxo estruturado no qual contratantes podem registrar demandas dentro do período de vigência do contrato, contratados realizam as entregas, e analistas de ponto de função avaliam essas entregas externamente, informando ao sistema a pontuação correspondente. Com base nessa pontuação e no valor por ponto de função previamente definido, o contrato inteligente realiza automaticamente a transferência do valor da carteira digital do contratante para a do contratado, promovendo um processo de pagamento eficiente, auditável e confiável.

6.1 Diagramas

Para representar de forma clara o funcionamento da solução proposta, esta seção apresenta os principais diagramas que descrevem a dinâmica do sistema e a interação entre seus componentes. São utilizados o diagrama de casos de uso, para fornecer uma visão geral das funcionalidades disponíveis aos diferentes perfis de usuário, e os diagramas de sequência, que detalham os fluxos específicos de criação de contrato e criação de demanda.

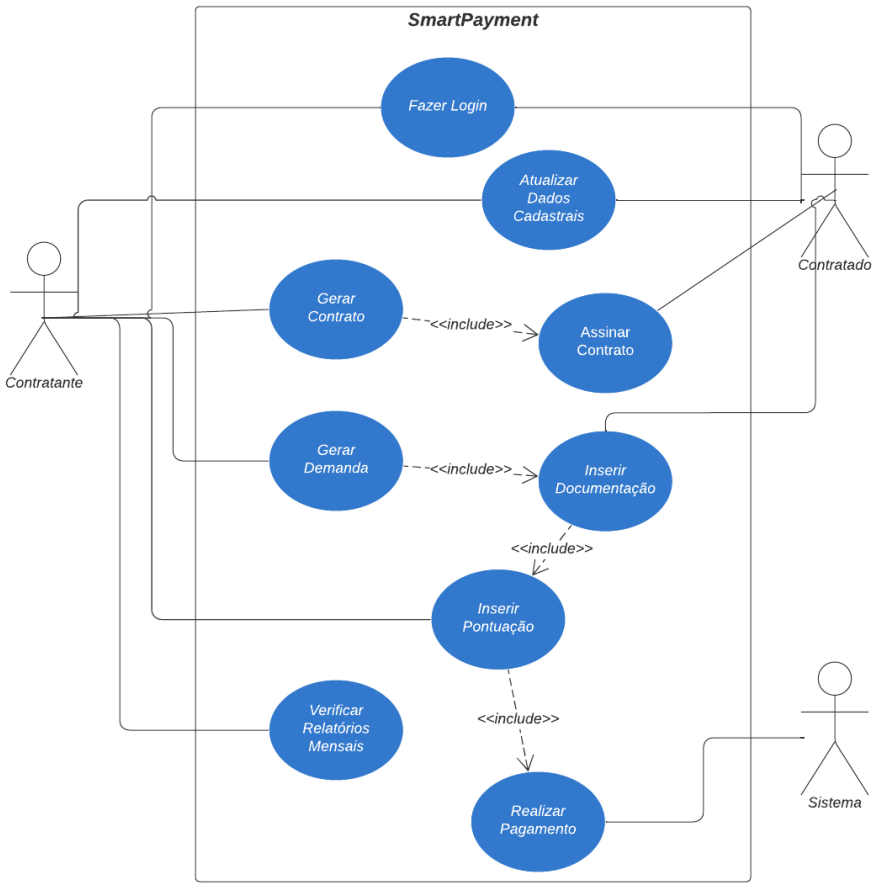
Essas representações visuais são fundamentais para demonstrar o comportamento dinâmico do sistema proposto, validando o modelo arquitetural adotado e reforçando a viabilidade da automação por meio de contratos inteligentes.

6.1.1 Diagrama de Casos de Uso

Esse diagrama documenta o que o sistema faz do ponto de vista do usuário. Em outras palavras, ele descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema. O diagrama é composto basicamente por quatro partes (RIBEIRO, 2012):

- **Cenário**: Sequência de eventos que acontecem quando um usuário interage com o sistema.
- **Ator**: Usuário do sistema, ou melhor, um tipo de usuário.
- **Casos de uso**: É uma tarefa ou uma funcionalidade realizada pelo ator (usuário).
- **Comunicação**: É o que liga um ator com um caso de uso

Figura 20 – Diagrama de Casos de Uso do sistema SmartPayment.



Fonte: Autor

O diagrama de caso de uso acima ilustra as interações entre os atores principais (Contratante, Contratado e o próprio Sistema) e as funcionalidades oferecidas pela aplicação. Os casos de uso contemplam desde o login e atualização de dados cadastrais até a geração de contratos e demandas, incluindo etapas como a inserção de documentação e pontuação.

O Contratante é responsável por iniciar os principais processos, como a criação de contratos, geração de demandas, inserção de pontuação e verificação de relatórios mensais. Além disso, é por meio da conta do contratante que são realizadas as avaliações das demandas e os respectivos pagamentos automatizados.

O Contratado, por sua vez, tem como atribuições a assinatura dos contratos e o envio da documentação relacionada às demandas recebidas.

O Sistema é o responsável por executar automaticamente os pagamentos, com base na pontuação atribuída às demandas, conforme os parâmetros definidos no contrato inteligente.

Este diagrama evidencia o fluxo de atividades e as dependências entre os atores envolvidos, fornecendo uma visão geral das funcionalidades do sistema SmartPayment e de como elas se conectam às responsabilidades de cada perfil de usuário.

6.1.2 Diagramas de Sequência

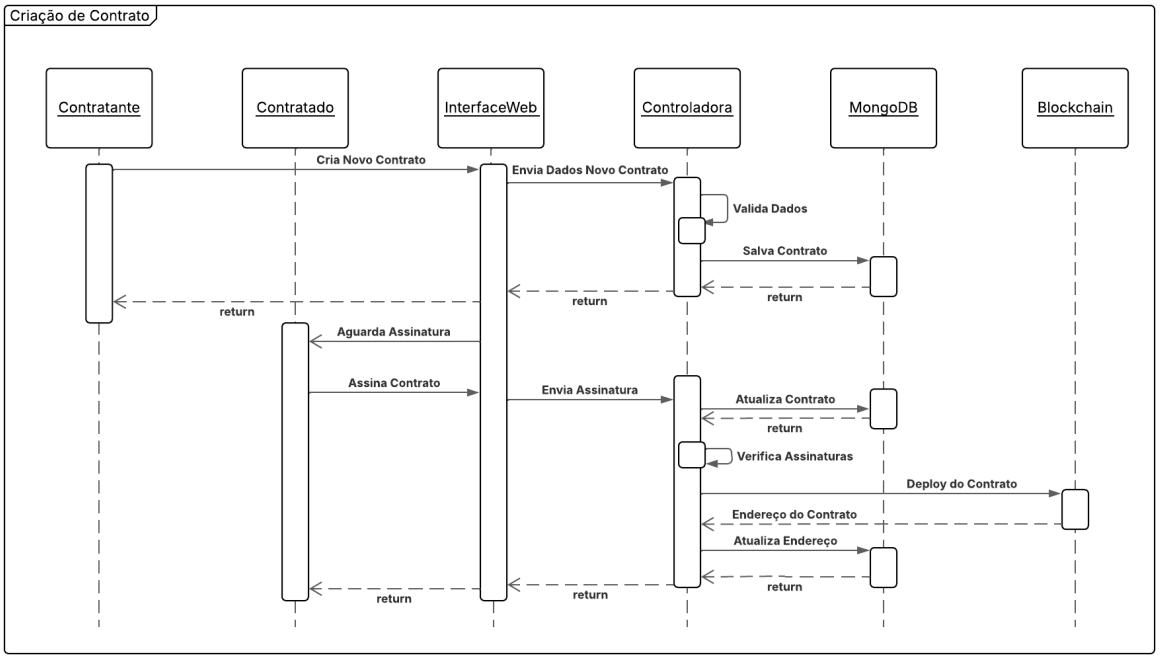
Os diagramas de sequência descrevem, em maior profundidade, a comunicação entre os componentes da aplicação durante dois processos principais: a criação de contratos e a criação de demandas. Esses diagramas mostram o fluxo de mensagens entre os usuários, a interface web, a controladora, o banco de dados e a blockchain, evidenciando as validações, persistência de dados e transações automatizadas que ocorrem no sistema.

6.1.2.1 Diagrama de Sequência da Criação de Contrato

Figura 21 apresenta o fluxo de criação de um contrato. Inicialmente, o contratante preenche os dados do novo contrato por meio da interface web, que os encaminha para a camada controladora. Esta valida as informações recebidas e armazena os dados no banco de dados, atribuindo ao contrato o status “Aguardando assinatura”.

Em seguida, o contratado acessa o contrato e, caso ainda não tenha assinado, pode confirmar sua participação. A assinatura é enviada pela interface à controladora, que atualiza o contrato no banco de dados. Se ambas as partes tiverem assinado, a controladora realiza a implantação do contrato na blockchain. O endereço gerado na blockchain é então atualizado no banco de dados e retornado ao sistema para exibição.

Figura 21 – Representação da Criação de um Contrato



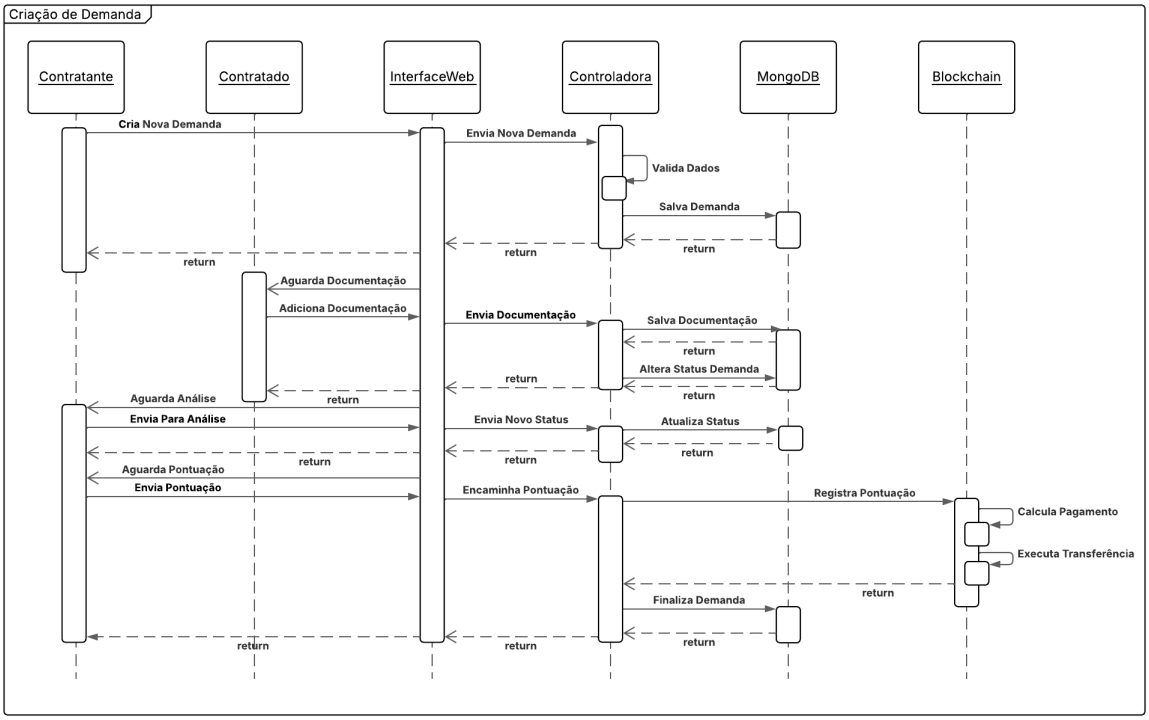
Fonte: Autor

6.1.2.2 Diagrama de Sequência da Criação de Demanda

A Figura 22 ilustra o processo de criação de uma nova demanda associada a um contrato. O contratante inicia o processo preenchendo os dados da demanda, que são validados pela camada controladora e salvos no banco de dados com o status "Aguardando documentação".

O contratado, então, insere a documentação referente à demanda. Após essa etapa, o contratante pode enviá-la para análise. A controladora atualiza o status para "Em análise", e o contratante fica apto a registrar a pontuação da demanda. Ao inserir a pontuação, a controladora registra a informação, calcula automaticamente o valor a ser pago com base na pontuação e no valor por ponto de função, e realiza a transferência de fundos via blockchain. Por fim, a demanda é marcada como finalizada no banco de dados.

Figura 22 – Representação da Criação de uma Demanda



Fonte: Autor

6.2 Linguagens e Ferramentas utilizadas para implementação do sistema SmartPayment

Para o desenvolvimento do sistema proposto, foram utilizadas as seguintes tecnologias:

6.2.1 Python

6.2.1.1 Backend

A linguagem Python foi utilizada tanto no backend quanto no frontend do sistema SmartPayment. No backend, ela é responsável pela lógica do servidor, incluindo o tratamento de requisições HTTP, autenticação de usuários, gerenciamento de sessões, comunicação com o banco de dados MongoDB e interação com a blockchain por meio da biblioteca Web3.py.

Um exemplo dessa implementação pode ser observado na Figura 23, que mostra o trecho de código responsável por receber os dados do contrato via requisição HTTP e armazená-los no banco de dados.

Figura 23 – Código em Python para criação do contrato - Backend.

contracts.py M

SmartPayment > app > routes > contracts.py > ...

You, last month | 1 author (You)

```python
from flask import Blueprint, request, jsonify, session
from app.repositories.user_repository import find_user_by_username, find_user_by_address
from app.blockchain import deploy_contract, enviar_pontuacao
from app.database import get_filtered_contracts, contracts_collection
from datetime import datetime
from bson import ObjectId

contracts_bp = Blueprint("contracts/api", __name__)


@contracts_bp.route("/criar_contrato", methods=["POST"])
def criar_contrato():
    data = request.get_json()
    required_fields = {"contratante", "contratado", "data_inicio", "data_fim", "valor_por_pf", "moeda"}

    if not required_fields.issubset(data):
        return jsonify({"error": "Campos obrigatórios ausentes"}), 400

    data_inicio_fmt = datetime.strptime(data["data_inicio"], "%Y-%m-%d").strftime("%d/%m/%Y")
    data_fim_fmt = datetime.strptime(data["data_fim"], "%Y-%m-%d").strftime("%d/%m/%Y")

    contrato = {
        "contratante": data["contratante"],
        "contratado": data["contratado"],
        "data_inicio": data_inicio_fmt,
        "data_fim": data_fim_fmt,
        "valor_por_pf": data["valor_por_pf"],
        "valor_inicial": data.get("valor_inicial", 0),
        "moeda": data["moeda"],
        "assinatura_contratante": data["assinatura_contratante"],
        "assinatura_contratado": data.get("assinatura_contratado", False),
        "status": "Aguardando assinatura",
        "contract_address": None
    }

    contracts_collection.insert_one(contrato)
    return jsonify({"message": "Contrato criado com sucesso."}), 201
```

Fonte: Autor

6.2.1.2 Frontend

No frontend, Python é utilizado através do framework Flask, que renderiza páginas HTML dinâmicas com o auxílio do mecanismo de templates Jinja2. Isso permite exibir dados do banco de forma organizada na interface do usuário, como contratos, demandas e relatórios.

A Figura 24 apresenta o trecho do código Python no frontend responsável por receber os dados preenchidos no formulário de criação de contrato e enviá-los para a API do backend. A utilização do Flask como estrutura central do sistema possibilitou uma integração fluida entre as camadas de visualização e lógica de negócio.

Figura 24 – Código em Python para criação do contrato - Frontend.

routes.py M

SmartPayment-Frontend > routes.py > ...

```python
@frontend_bp.route("/novo_contrato", methods=["GET", "POST"])
def novo_contrato():
    if request.method == "POST":
        try:
            username = session.get("user")
            if not username:
                return redirect(url_for("frontend.login"))

            user_response = requests.get(f"{BACKEND_URL}/user/info", params={"username": username})
            if user_response.status_code != 200:
                return render_template("contratos/novo_contrato.html", error="Não foi possível obter sua conta.")

            contratante = user_response.json().get("conta")

            # Coletar dados do formulário
            contratado = request.form.get("contratado")
            data_inicio = request.form.get("data_inicio")
            data_fim = request.form.get("data_fim")
            valor_pf = int(request.form.get("valor_pf"))
            valor_inicial = 0.0

            moeda = request.form.get("moeda")

            # Montar payload
            payload = {
                "contratante": contratante,
                "contratado": contratado,
                "valor_por_pf": valor_pf,
                "valor_inicial": valor_inicial,
                "data_inicio": data_inicio,
                "data_fim": data_fim,
                "assinatura_contratante": True,
                "assinatura_contratado": False,
                "status": "Aguardando assinatura",
                "moeda": moeda
            }

            print("📤 Enviando payload para backend:", payload)

            # Enviar para o backend
            response = requests.post(f"{BACKEND_URL}/contracts/criar_contrato", json=payload)

            if response.status_code in [200, 201]:
                result = response.json()
                return render_template("contratos/novo_contrato.html", success=True, result=result)
            else:
                try:
                    erro_backend = response.json().get("error", "Erro desconhecido.")
                except:
                    erro_backend = response.text  # fallback
                return render_template("contratos/novo_contrato.html", error=f"Erro do backend: {erro_backend}")

        except Exception as e:
            return render_template("contratos/novo_contrato.html", error=f"Erro: {e}")

    return render_template("contratos/novo_contrato.html")
```

Fonte: Autor

6.2.2 Solidity

A linguagem Solidity foi utilizada para o desenvolvimento dos contratos inteligentes que automatizam os pagamentos no sistema SmartPayment. O contrato principal define as regras do relacionamento entre contratante e contratado, incluindo datas, valores e permissões, além de permitir o uso de diferentes moedas digitais como ETH, USDC e Drex.

A Figura 25 apresenta trechos fundamentais do código do contrato inteligente, destacando o construtor, responsável por inicializar o contrato com os dados essenciais como os endereços das partes, datas de vigência, valor por ponto de função e o token

utilizado. Também são mostradas as funções:

- `ativarContrato`: que permite ao contratante ativar o contrato na data definida, desde que ele ainda esteja pendente;
- `registrarPontuacao`: utilizada para calcular o valor a ser pago com base na pontuação enviada e realizar o pagamento automaticamente, seja em ETH ou via token ERC-20;
- `finalizarContrato`: que muda o estado do contrato para concluído, quando a data de fim é atingida.

Figura 25 – Trecho do código em Solidity para criação do contrato inteligente.

SmartPayment.sol M

smartContract > contracts > SmartPayment.sol > ...

```
5

48
49      constructor(address _contratado, uint _dataInicio, uint _dataFim, uint _valorPorPF, address _token) {
50          require(_dataFim > _dataInicio, "Data de fim deve ser posterior a data de inicio.");
51
52          contratante = msg.sender;
53          contratado = _contratado;
54          dataInicio = _dataInicio;
55          dataFim = _dataFim;
56          valorPorPF = _valorPorPF;
57          token = IERC20(_token);
58          statusContrato = StatusContrato.Pendente;
59
60          emit ContratoCriado(contratante, contratado, dataInicio, dataFim, valorPorPF);
61      }
62
63      function ativarContrato() public apenasContratante {
64          require(block.timestamp >= dataInicio, "Contrato nao pode ser ativado antes da data de inicio.");
65          require(statusContrato == StatusContrato.Pendente, "Contrato ja esta ativo ou concluido.");
66
67          statusContrato = StatusContrato.Ativo;
68          emit ContratoAtivado(block.timestamp);
69      }
70
71      function registrarPontuacao(uint pontos) public payable apenasContratante contratoAtivo {
72          uint valorPago = pontos * valorPorPF;
73          require(valorPago > 0, "Valor do pagamento deve ser maior que zero.");
74
75          totalPontos += pontos;
76          totalPago += valorPago;
77
78          if (address(token) == address(0)) {
79              // Pagamento em ETH
80              require(msg.value == valorPago, "Valor enviado incorreto.");
81              payable(contratado).transfer(valorPago);
82          } else {
83              // Pagamento via token
84              require(msg.value == 0, "Nao envie ETH ao usar token.");
85              require(token.allowance(contratante, address(this)) >= valorPago, "Contrato nao aprovado para gastar token.");
86              require(token.balanceOf(contratante) >= valorPago, "Saldo insuficiente do contratante.");
87              token.transferFrom(contratante, contratado, valorPago);
88      }
89
90          emit PagamentoRealizado(contratado, pontos, valorPago);
91      }
92
93      function finalizarContrato() public apenasContratante {
94          require(block.timestamp >= dataFim, "Contrato ainda nao atingiu a data final.");
95          require(statusContrato == StatusContrato.Ativo, "Contrato deve estar ativo para ser finalizado.");
96
97          statusContrato = StatusContrato.Concluido;
98          emit ContratoFinalizado(block.timestamp);
99      }
100 }
```

Fonte: Autor

Essas funções foram projetadas com segurança e controle de acesso, utilizando modificadores personalizados que garantem que apenas o contratante autorizado possa executá-las e que as ações ocorram no tempo apropriado. A adoção do padrão ERC-20 para os tokens utilizados permite compatibilidade com diversas moedas digitais.

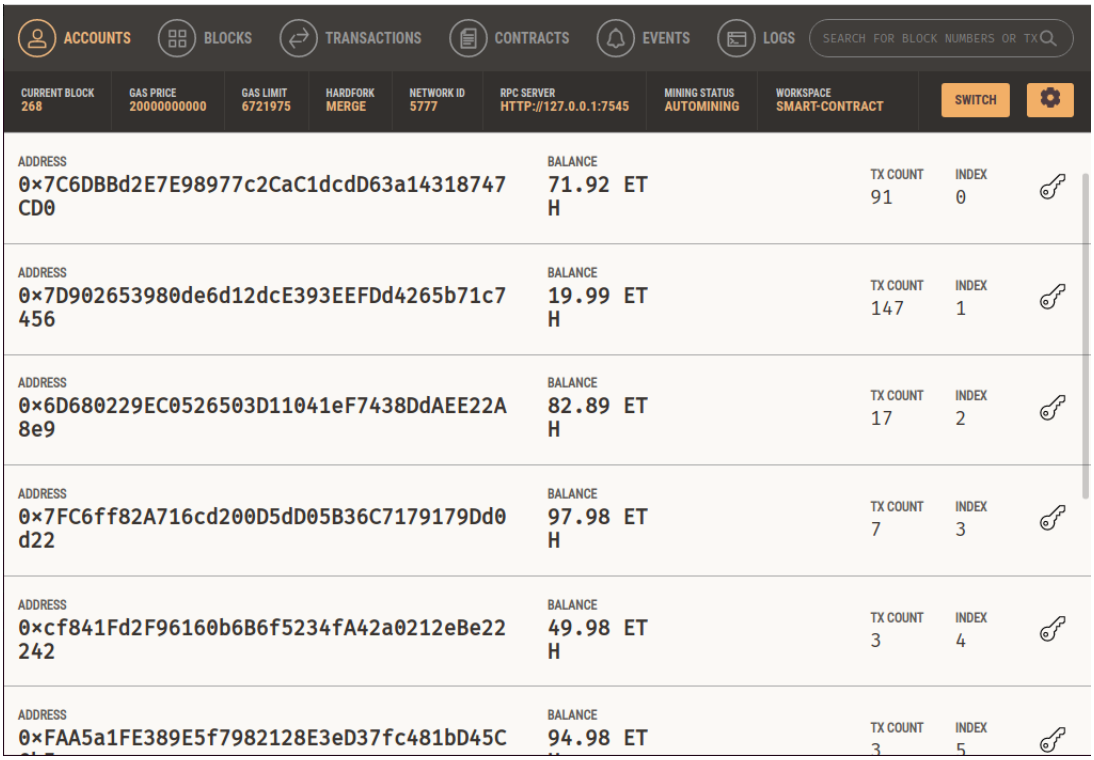
6.2.3 Ganache

A ferramenta Ganache foi utilizada como blockchain local para desenvolvimento e testes dos contratos inteligentes. Ela permite a simulação de uma rede Ethereum privada, com diversas contas pré-configuradas, permitindo implantar contratos inteligentes e realizar transações de maneira rápida e segura, sem a necessidade de acessar uma rede pública ou gastar ativos reais.

Na Figura 26, observa-se a interface da Ganache, que exibe informações detalhadas das contas simuladas, como seus endereços, saldos em ether (ETH), número de transações e a possibilidade de visualizar as chaves privadas. Cada uma dessas contas foi utilizada para representar diferentes atores do sistema, como contratante, contratado e o próprio contrato inteligente, durante os testes.

O uso da Ganache foi essencial para validar todo o fluxo de criação de contratos, envio de demandas e processamento automático dos pagamentos, garantindo a robustez do sistema antes de qualquer possível migração para uma rede de produção.

Figura 26 – Interface da Ganache exibindo contas e saldos simulados em rede blockchain local.



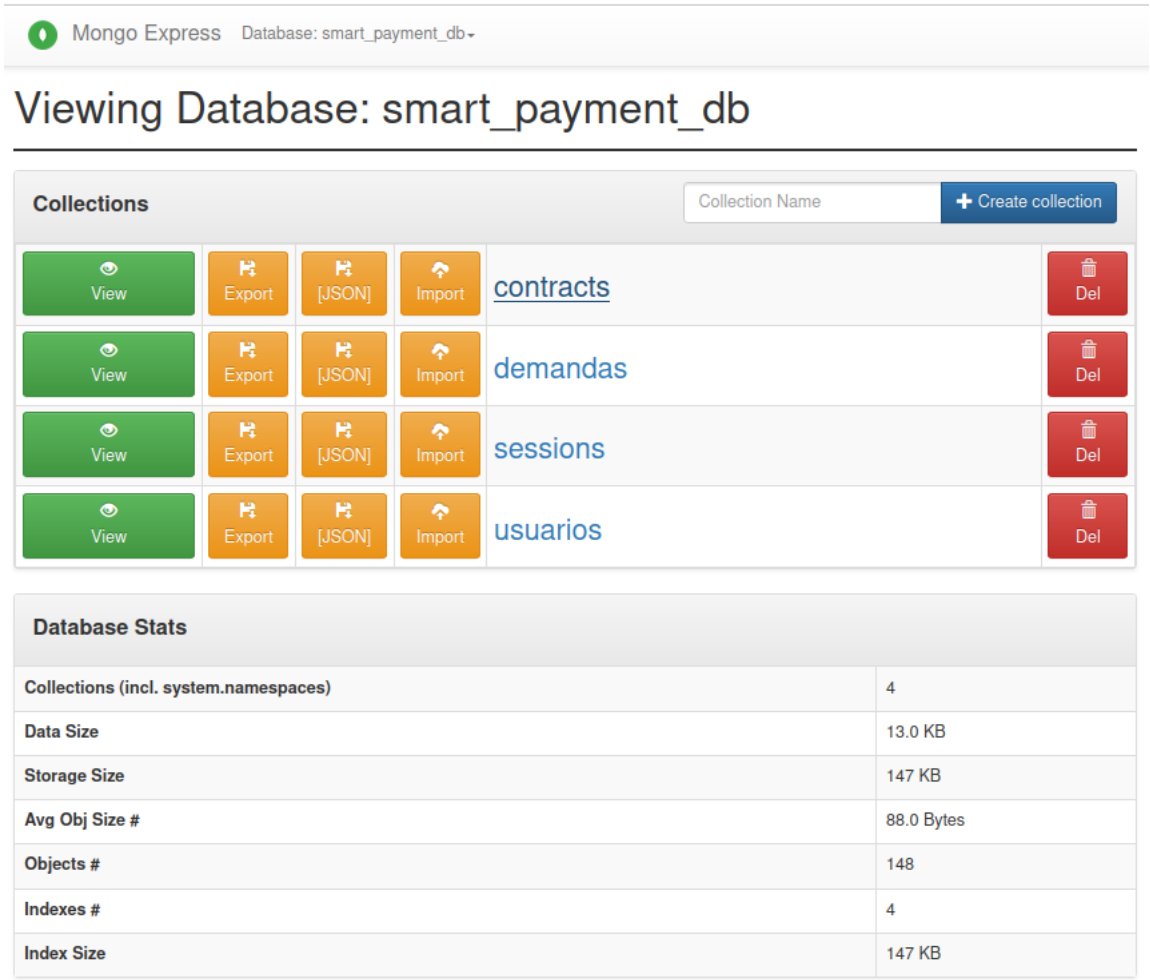
Fonte: Autor

6.2.4 MongoDB

O MongoDB foi utilizado como banco de dados principal do sistema SmartPayment, oferecendo uma estrutura flexível e orientada a documentos para armazenar as informações do sistema. Sua estrutura dinâmica permitiu armazenar informações variadas, como contratos, demandas, usuários e sessões de login, sem a necessidade de esquemas rígidos. A integração com o backend em Python foi realizada por meio da biblioteca PyMongo, permitindo a leitura, escrita e atualização de documentos diretamente nas coleções do banco.

A Figura 27 apresenta a interface da ferramenta Mongo Express, utilizada durante o desenvolvimento para inspecionar as coleções armazenadas na base de dados `smart_payment_db`. É possível observar visualmente as coleções contracts, demandas, sessions e usuarios, cada uma com papel específico na persistência das informações do sistema. A visualização por meio do Mongo Express facilitou a verificação de dados durante os testes e auxiliou na organização das estruturas ao longo da implementação.

Figura 27 – Coleções do banco de dados `smart_payment_db` exibidas na interface do Mongo Express.



Fonte: Autor

6.2.5 Javascript

A linguagem JavaScript foi utilizada no frontend do sistema SmartPayment para tornar a interface do usuário mais dinâmica e interativa. Scripts em JavaScript foram responsáveis por funcionalidades como exibição de pop-ups, controle de filtros, manipulação de elementos na tela e melhoria da experiência de navegação. O código JavaScript foi integrado às páginas HTML renderizadas pelo Flask, permitindo a atualização dinâmica de conteúdo e a interação fluida entre o usuário e os dados apresentados. O uso de JavaScript puro, sem frameworks adicionais, foi suficiente para atender às necessidades de interatividade do sistema.

Um exemplo pode ser observado na Figura 28, onde é implementada a função `showToast()`, responsável por exibir notificações temporárias (pop-ups) com diferentes tipos de alerta, como mensagens de sucesso ou erro.

Figura 28 – Código em JavaScript para exibição de pop-up.

pop-up.js

SmartPayment-Frontend > static > js > pop-up.js > ...

You, 1 second ago | 1 author (You)

```
function showToast(message, type = 'success') {
    const toast = document.getElementById("toast");
    const messageBox = document.getElementById("toast-message");

    messageBox.textContent = message;
    toast.className = `toast ${type}`;
    toast.classList.remove("hidden");

    setTimeout(() => {
        toast.classList.add("hidden");
    }, 5000);
}
```

Fonte: Autor

6.2.6 HTML e CSS

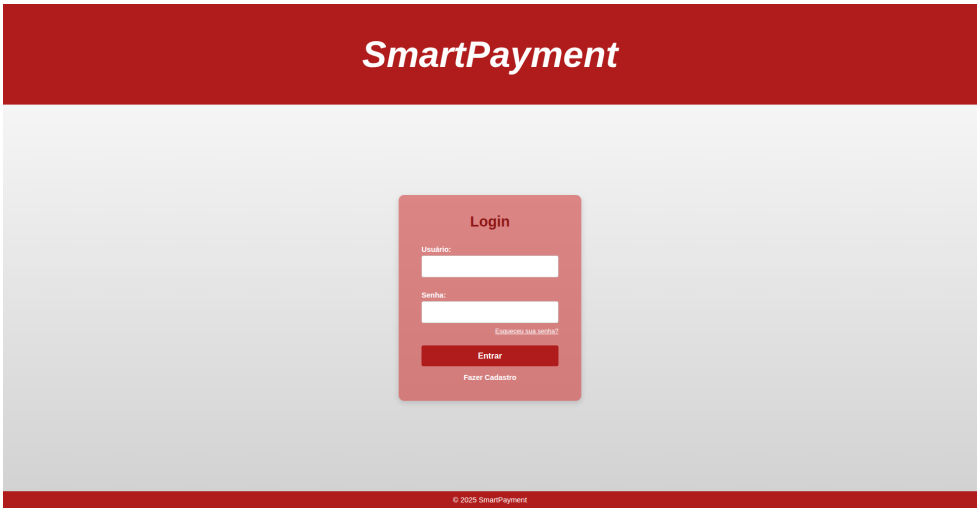
As páginas do sistema SmartPayment foram desenvolvidas utilizando HTML e CSS, responsáveis pela estruturação e estilização da interface do usuário. O HTML foi empregado para definir a organização dos elementos visuais nas páginas, como formulários, tabelas, botões e seções de navegação. Já o CSS foi utilizado para aplicar estilos e tornar a interface mais agradável e intuitiva, utilizando classes e regras para controle de cores, espaçamento, alinhamento e responsividade. As páginas desenvolvidas podem ser visualizadas na Seção 6.3, apresentada a seguir, onde são exibidas as telas da aplicação web implementada.

6.3 Telas

A seguir, são apresentadas as telas do sistema SmartPayment, capturadas diretamente da aplicação web desenvolvida.

6.3.1 Login

Figura 29 – Tela de Login.



Fonte: Autor

6.3.2 Contratos

A aplicação disponibiliza aos usuários uma interface completa para o gerenciamento de contratos inteligentes. A Figura 30 apresenta a tela de listagem de contratos, na qual o usuário pode visualizar todos os contratos nos quais está envolvido, seja como contratante ou contratado. As informações exibidas incluem o endereço do contrato, status, moeda utilizada, data de início e fim, além da possibilidade de acessar os detalhes de cada contrato individualmente.

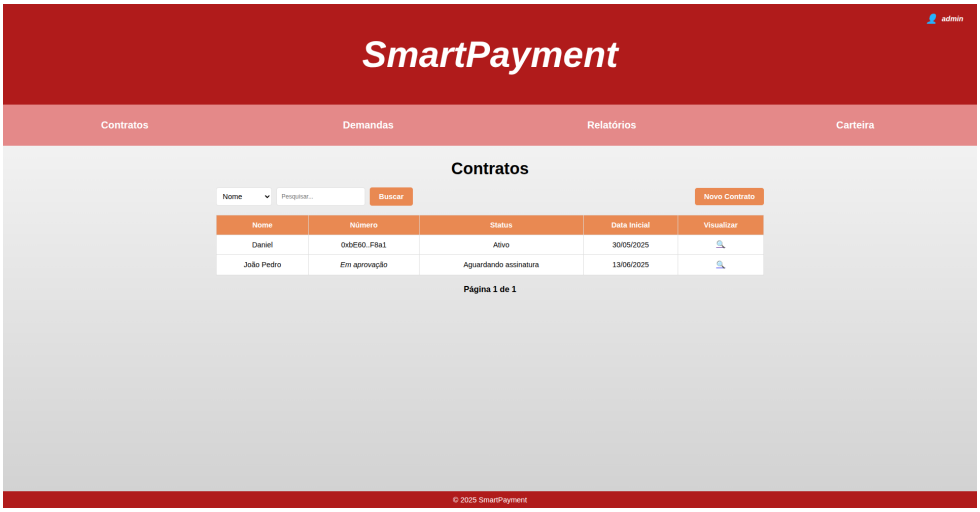
A Figura 31 exibe a tela de criação de um novo contrato, acessível apenas ao contratante. Nessa interface, o usuário informa os dados essenciais do contrato, como endereço do contratado, datas de vigência, valor por ponto de função e moeda utilizada para pagamento. Após o preenchimento, o contrato é registrado no sistema com status "Aguardando assinatura".

A Figura 32 mostra a tela de visualização dos detalhes de um contrato. Nessa interface, o usuário pode consultar todas as informações inseridas no momento da criação do contrato, bem como o status atual, assinatura dos participantes, moeda escolhida, entre outros dados.

Por fim, a Figura 33 representa a mesma tela de detalhes (Figura 32), porém exibida para o contratado que ainda não assinou o contrato. Nesse caso, é apresentado o botão "Assinar Contrato", permitindo ao contratado registrar sua assinatura e prosseguir com o processo de validação do contrato.

6.3.2.1 Lista de Contratos

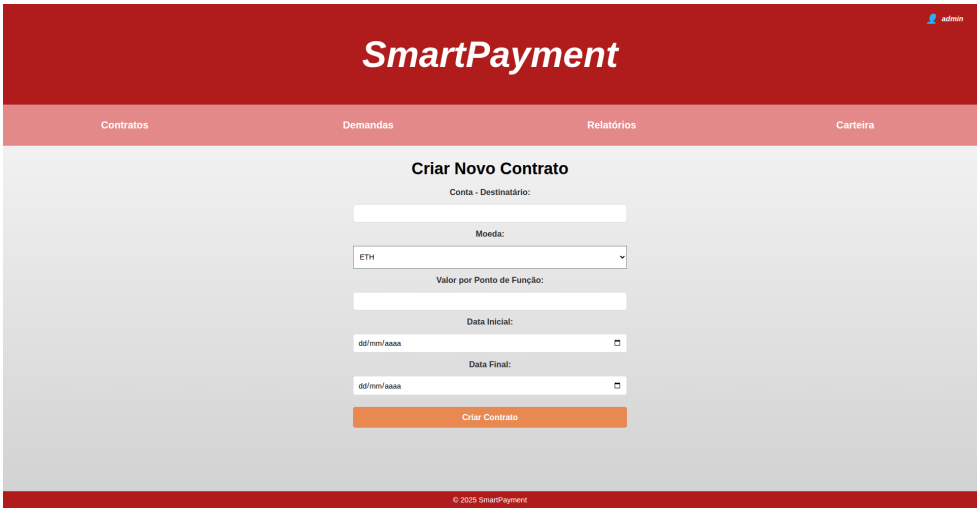
Figura 30 – Tela com relação de todos contratos ativos do usuário.



Fonte: Autor

6.3.2.2 Novo Contrato

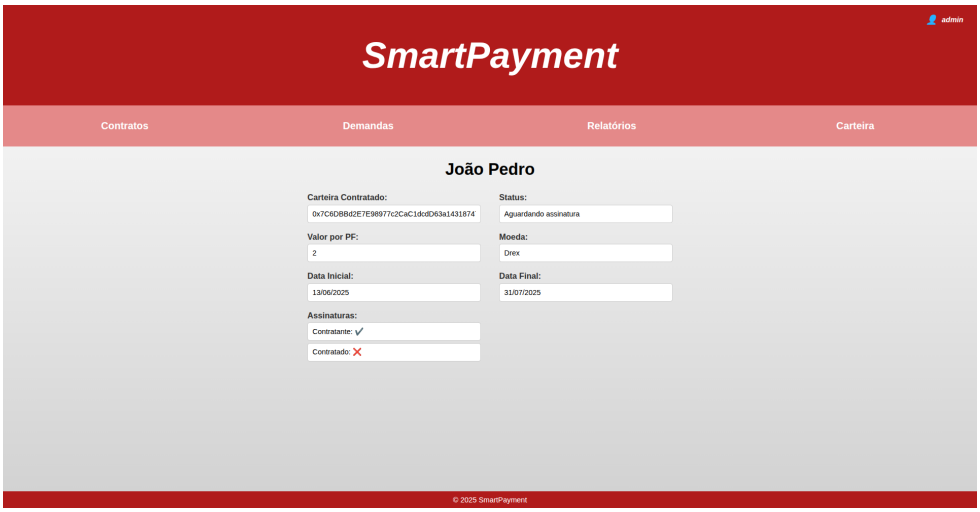
Figura 31 – Tela de criação de um novo contratos.



Fonte: Autor

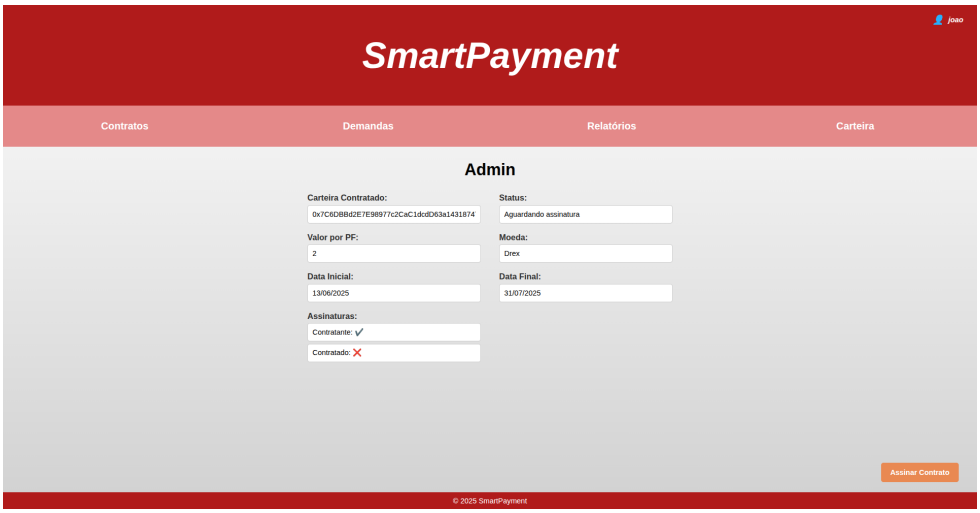
6.3.2.3 Detalhes do Contrato

Figura 32 – Tela com detalhes de um contrato gerado.



Fonte: Autor

Figura 33 – Tela do contratado com detalhes de um contrato aguardando assinatura.



Fonte: Autor

6.3.3 Demandas

Além dos contratos, os usuários também podem acompanhar e interagir com as demandas vinculadas a cada contrato em que participam. A figura 34 apresenta a tela de listagem de demandas, na qual o usuário pode visualizar todas as demandas nas quais está envolvido. As colunas exibidas incluem o título da demanda, participante envolvido, status atual, data de criação e um botão para acessar os detalhes da demanda.

A Figura 35 exibe a tela de criação de novas demandas. Nessa interface, o usuário informa o título da demanda, seleciona o contrato associado e define uma breve descrição

do que será entregue. Após o envio, a demanda é registrada com o status "Aguardando documentação".

A Figura 36 mostra a página acessada pelo contratado, onde ele pode anexar a documentação correspondente à demanda. Essa tela disponibiliza um campo para upload do arquivo e o botão “Salvar Documentação”, que atualiza o sistema com o material fornecido.

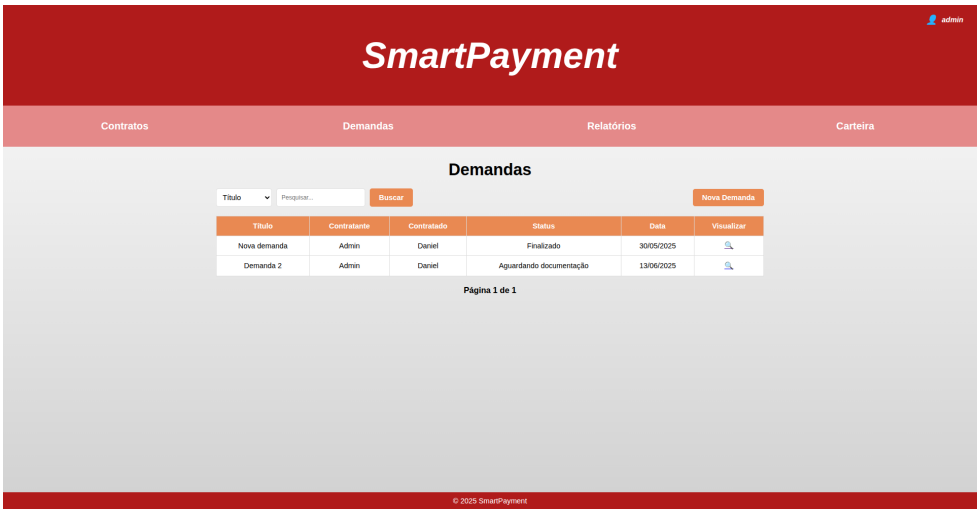
Após o contratado salvar a documentação, a tela evolui conforme mostrado na Figura 37. Se o usuário logado for o contratante, é exibido o botão “Enviar para análise”, permitindo que a demanda avance para a próxima etapa do fluxo.

A Figura 38 apresenta a tela seguinte, acessada também pelo contratante, na qual é possível inserir a pontuação de Pontos de Função (PF) atribuída à demanda. Essa pontuação será utilizada para calcular o valor a ser pago automaticamente pelo contrato inteligente.

Por fim, a Figura 39 mostra a tela final da demanda, quando todas as etapas já foram concluídas e o status está marcado como "Finalizada". Nessa interface, são exibidos apenas os dados informativos da demanda, sem mais opções de ação, refletindo a conclusão do processo.

6.3.3.1 Lista de Demandas

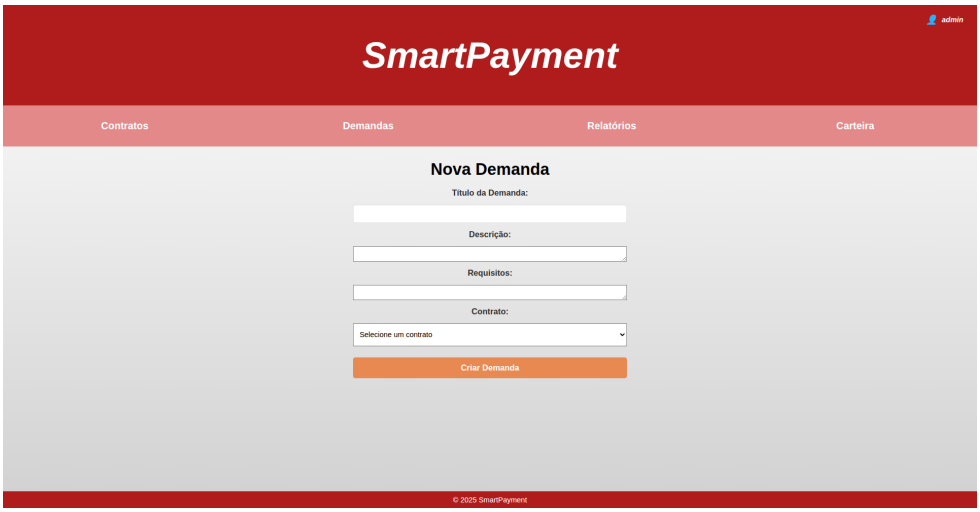
Figura 34 – Tela com relação de todas demandas ativas de um usuário.



Fonte: Autor

6.3.3.2 Nova Demanda

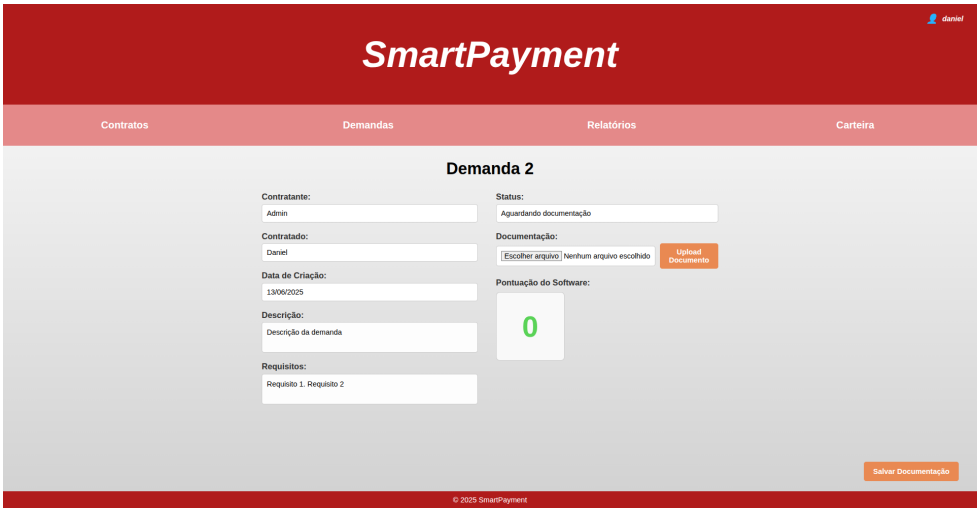
Figura 35 – Tela de criação de uma nova demanda.



Fonte: Autor

6.3.3.3 Salvar Documentação

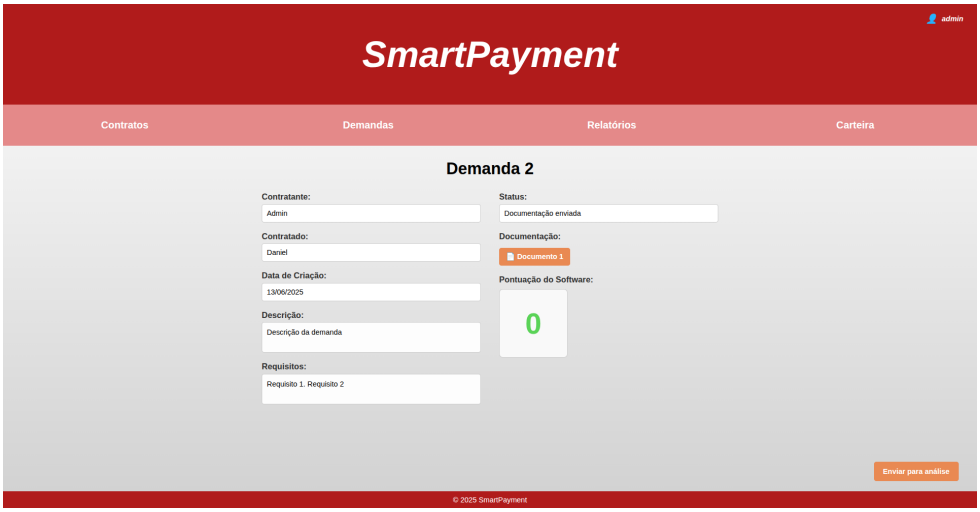
Figura 36 – Tela com detalhes da demanda previamente a inserção da documentação.



Fonte: Autor

6.3.3.4 Enviar Demanda para Análise

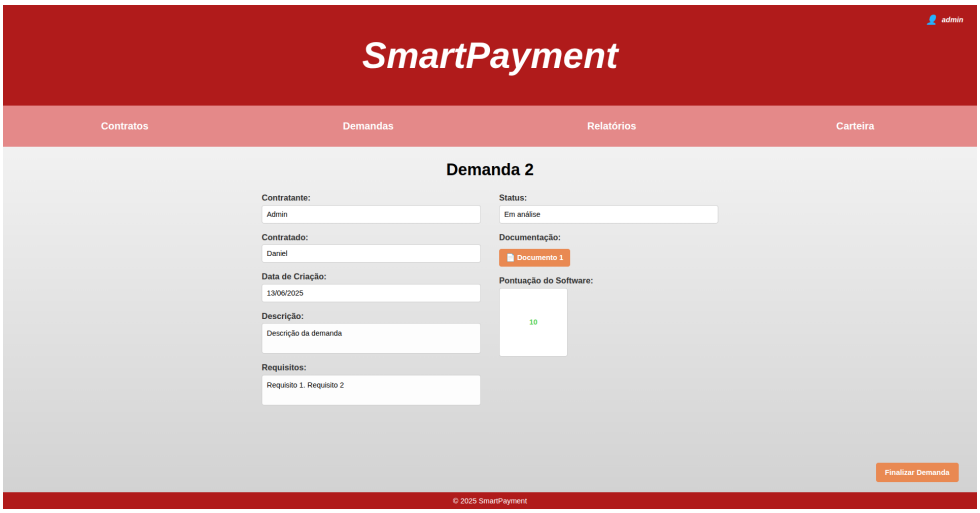
Figura 37 – Tela com detalhes da demanda previamente ao envio para análise.



Fonte: Autor

6.3.3.5 Inserir Pontuação da Demanda

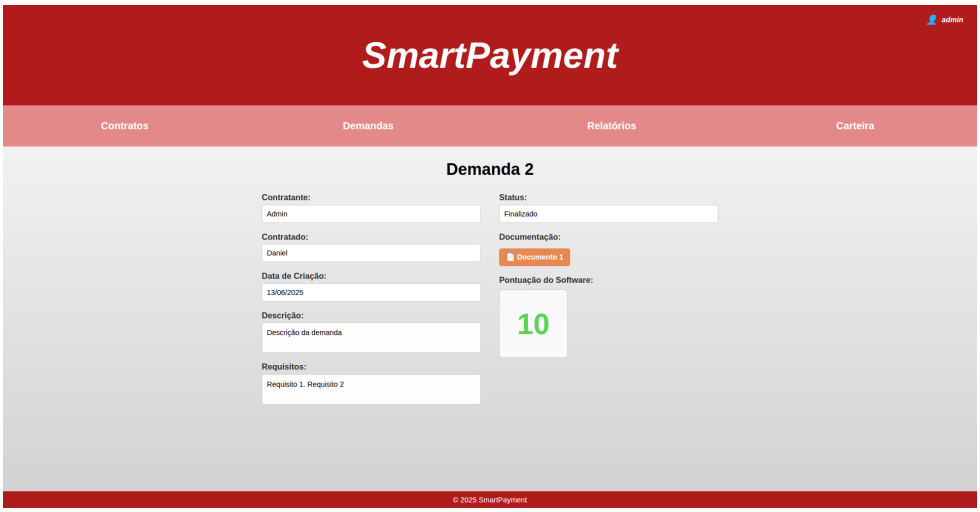
Figura 38 – Tela com detalhes da demanda previamente a inserção da pontuação.



Fonte: Autor

6.3.3.6 Demanda Finalizada

Figura 39 – Tela com detalhes da demanda finalizada.



Fonte: Autor

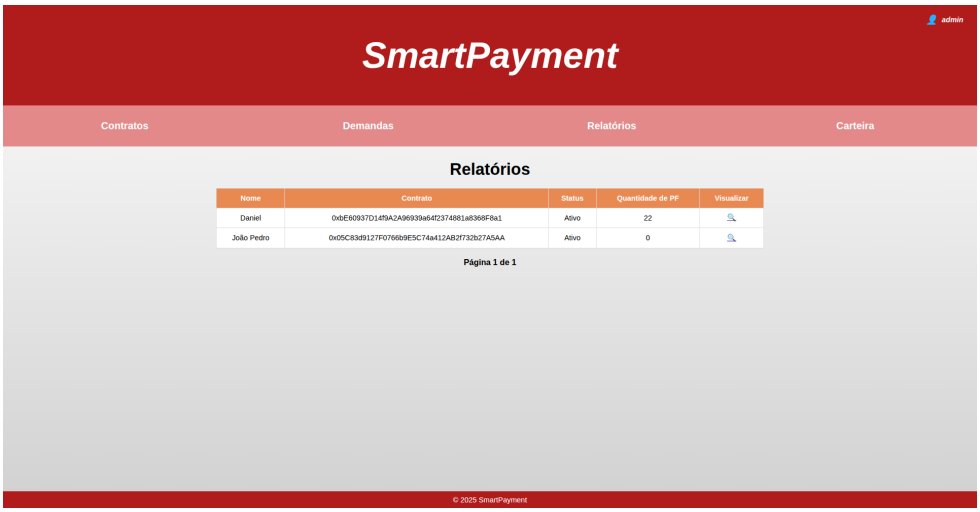
6.3.4 Relatórios

A funcionalidade de relatórios permite ao usuário acompanhar, de forma consolidada, os resultados obtidos a partir das demandas registradas em cada contrato. A Figura 40 apresenta a tela de listagem de contratos com relatório associado. Nessa interface, o sistema exibe todos os contratos nos quais o usuário está envolvido, juntamente com o somatório da pontuação total acumulada em todas as demandas vinculadas a cada contrato. Essa visão facilita a análise do desempenho geral do contrato ao longo de sua vigência.

A Figura 41 ilustra a tela de detalhamento do relatório mensal. Após a seleção de um contrato e de um mês específico, são exibidas todas as demandas associadas àquele período, com informações como título da demanda, status e pontuação atribuída. A interface também apresenta o somatório da pontuação referente apenas às demandas do mês selecionado, fornecendo uma visão segmentada do desempenho mensal do contrato.

6.3.4.1 Lista de Relatórios

Figura 40 – Tela com relação da quantidade de PFs entregues por contrato.



Fonte: Autor

6.3.4.2 Lista de Demandas Mensais

Figura 41 – Tela com todas demandas a uma empresa em um determinado mês.

Fonte: Autor

6.3.5 Dados Cadastrais

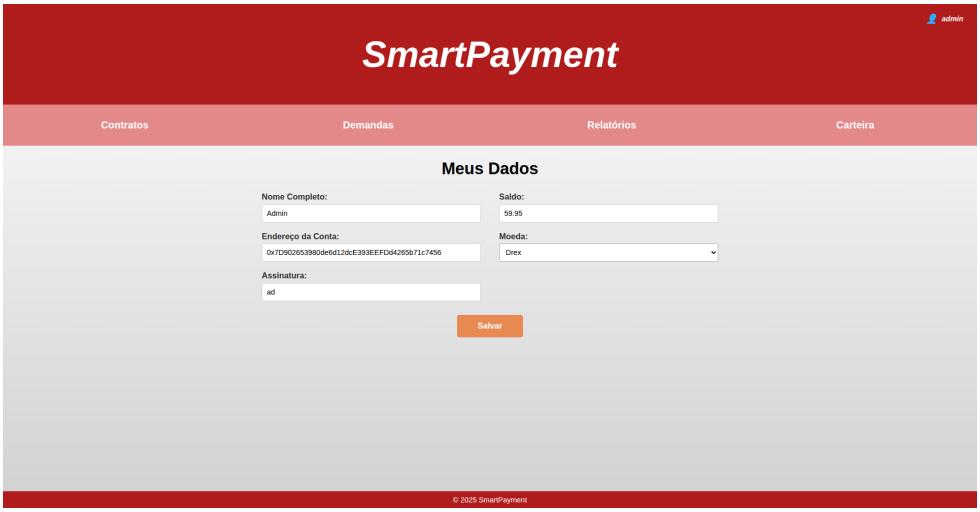
A Figura 42 apresenta a tela de dados cadastrais do usuário, onde são exibidas as informações essenciais vinculadas à sua conta. Essa interface foi projetada para oferecer uma visualização clara dos dados pessoais e financeiros do usuário dentro da aplicação.

Os campos apresentados incluem o nome completo, o endereço da conta na blockchain, a assinatura digital e o saldo disponível. A tela também contém um campo de seleção de moeda, permitindo ao usuário alternar entre as moedas suportadas pelo sis-

tema (como ETH, Drex ou USDC). O valor exibido no campo de saldo é ajustado dinamicamente de acordo com a moeda selecionada, refletindo o saldo correspondente àquela unidade monetária.

Essa tela oferece ao usuário uma visão consolidada de sua identidade no sistema, além de informações atualizadas sobre seus recursos financeiros disponíveis para interações com contratos e demandas.

Figura 42 – Tela com os dados cadastrais do usuário.



Fonte: Autor

7 Considerações Finais

7.1 Conclusão

O presente trabalho teve como objetivo o desenvolvimento de uma aplicação web capaz de automatizar a gestão e a compensação financeira em projetos de desenvolvimento de software, por meio da utilização de contratos inteligentes. A proposta consistiu em criar um sistema no qual contratos e demandas pudessem ser gerenciados diretamente entre contratantes e contratados, sendo que, após a entrega de cada demanda, um analista de pontos de função realizaria a avaliação e registraria a respectiva pontuação. Com base nessa pontuação e nos parâmetros estabelecidos no contrato, os pagamentos seriam processados automaticamente pelo sistema, por meio da tecnologia de contratos inteligentes, sem a necessidade de intervenção de intermediários.

A escolha do tema surgiu a partir de uma demanda real apresentada aos autores deste trabalho, na qual foi relatado um processo comum em empresas que contratam fábricas de software para realizar entregas periódicas. Atualmente, nesse processo, após a entrega das demandas, é realizada uma análise de ponto de função por um analista da empresa contratante. Em seguida, a pontuação gerada é repassada à área contábil, que se responsabiliza pelo pagamento. Assim, o sistema proposto buscou centralizar esse fluxo, eliminando a necessidade de repassar informações entre diferentes setores e promovendo maior agilidade, confiabilidade e transparência nos pagamentos.

O desenvolvimento da aplicação foi concluído com sucesso, atendendo integralmente aos objetivos definidos. O sistema implementado possibilita a criação e o gerenciamento de contratos e demandas, o registro de pontuações por analistas de ponto de função e a execução automática dos pagamentos. Além disso, foi projetado para ser compatível com o Drex, a moeda digital brasileira, assegurando sua viabilidade futura em um cenário de adoção dessa tecnologia.

Este trabalho também representou uma oportunidade ímpar de aplicação prática dos conhecimentos adquiridos ao longo do curso de Engenharia de Software, integrando conceitos teóricos e técnicos em um tema diretamente relacionado à realidade dos autores. O projeto possibilitou uma conexão de áreas relevantes do mercado de software, como contratos inteligentes, criptomoedas, tecnologias web, usabilidade, banco de dados e blockchain, resultando em uma solução funcional, inovadora e de impacto prático.

7.2 Trabalhos Futuros

Embora o sistema SmartPayment tenha sido desenvolvido com foco na entrega de todas as funcionalidades propostas, algumas melhorias podem ser consideradas para versões futuras. Uma das possibilidades é a realização de ajustes estruturais no código-fonte, com o objetivo de aprimorar sua organização interna, modularidade e aderência a padrões consolidados de engenharia de software. Essa refatoração contribuiria para facilitar a manutenção do sistema e a implementação de novos recursos de forma mais eficiente.

Outra sugestão é a integração com sistemas de autenticação baseados em identidade digital descentralizada (DID), ampliando a segurança e a rastreabilidade das ações dos usuários na plataforma. Também se vislumbra a possibilidade de suporte nativo a carteiras digitais externas, permitindo maior flexibilidade para os usuários na gestão dos ativos utilizados nos contratos inteligentes.

Adicionalmente, à medida que o Drex for oficialmente disponibilizado e regulamentado, poderão ser realizados testes em ambientes públicos ou redes de testes oficiais, substituindo a rede local de desenvolvimento utilizada no projeto. Isso permitirá validar a compatibilidade do sistema com os padrões adotados pelo Banco Central do Brasil para a moeda digital.

A qualidade de software é um aspecto relevante para o sucesso e a evolução do sistema SmartPayment, especialmente no que diz respeito à experiência e satisfação dos usuários. Por essa razão, recomenda-se que trabalhos futuros incluam a realização de testes voltados à avaliação da qualidade do sistema. Em particular, é importante implementar testes de usabilidade, de adequação funcional, de efetividade e de satisfação, conforme descrito nas Tabelas 2 e 3, presentes no Capítulo 4. Esses testes permitirão identificar pontos de melhoria sob a perspectiva dos usuários e contribuirão para garantir que a solução atenda, de forma prática, eficiente e acessível, às necessidades reais do seu público-alvo.

Essas melhorias visam não apenas evoluir a aplicação, mas também mantê-la alinhada com as tendências tecnológicas e regulatórias do mercado de blockchain e contratos inteligentes.

Referências

BANCO Central do Brasil. 2022. Acessado em 16 de agosto de 2024. Disponível em: <https://www.bcb.gov.br/estabilidadefinanceira/drex>. Citado 2 vezes nas páginas 13 e 35.

BANCO Central do Brasil. 2023. Acessado em 16 de agosto de 2024. Disponível em: <https://www.bcb.gov.br/meubc/faqs/p/lancamento-do-drex>. Citado na página 36.

BINANCE. 2024. Acessado em 16 de agosto de 2024. Disponível em: <https://www.binance.com/en/price/bitcoin>. Citado na página 34.

CAVALCANTI, M. O. de M.; NóBREGA, M. Smart contracts ou "contratos inteligentes": o direito na era da blockchain. *Revista Científica Disruptiva*, v. 2, n. 1, p. 96, 2020. Citado na página 37.

DIFFIE, W.; HELLMAN, M. Exhaustive cryptanalysis of the nbs data encryption standard. *Computer*, junho 1977. Citado na página 21.

FORBES. 2022. Acessado em 16 de agosto de 2024. Disponível em: <https://forbes.com.br/forbes-money/2022/02/forbes-top-50-blockchain-conheca-as-empresas-bilionarias-que-utilizam-a-tecnologia/>. Citado 2 vezes nas páginas 32 e 33.

FREEPIK. 2024. Acessado em 16 de agosto de 2024. Disponível em: <https://br.freepik.com/fotos-gratis/renderizacao-3d-da-tecnologia-blockchain_196469723.htm#query=tecnologia%20blockchain&position=4&from_view=keyword&track=ais_hybrid&uuid=0858fba9-0594-4c29-a876-21db3eacf9da>. Citado na página 31.

HARTIKKA, L. *Medium*. 2017. Acessado em 16 de agosto de 2024. Disponível em: <https://medium.com/@lhartikk/a-blockchain-in-200-lines-of-code-963cc1cc0e54>. Citado na página 31.

IBM. 2024. Acessado em 16 de agosto de 2024. Disponível em: <https://www.ibm.com/br-pt/topics/blockchain>. Citado na página 30.

INFOMONEY. 2024. Acessado em 16 de agosto de 2024. Disponível em: <https://www.infomoney.com.br/minhas-financas/o-que-esperar-do-drex-em-2024/>. Citado na página 35.

ISO/IEC. *ISO/IEC 25010:2011: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. Geneva, 2011. Geneva: ISO. Citado 3 vezes nas páginas 38, 39 e 40.

ISO/IEC 20926:2009. 2009. Acessado em 16 de agosto de 2024. Disponível em: <https://www.iso.org/standard/51717.html>. Citado na página 54.

ISO/IEC 24570:2018. 2018. Acessado em 16 de agosto de 2024. Disponível em: <https://www.iso.org/standard/72505.html>. Citado na página 54.

KISSEL, R. *Glossary of Key Information Security Terms*. [S.l.], 2019. Disponível em: <https://nvlpubs.nist.gov/nistpubs/ir/2013/nist.ir.7298r2.pdf>. Citado na página 20.

MALAR, J. P. *Exame*. 2023. Acessado em 16 de agosto de 2024. Disponível em: <https://exame.com/future-of-money/banco-central-europeu-reconhece-bitcoin-reserva-valor/>. Citado na página 34.

NESMA. 2019. Acessado em 16 de agosto de 2024. Disponível em: <https://nesma.org/wp-content/uploads/2015/07/FPA-according-to-Nesma-and-IFPUG-vs-2019-01-17.pdf>. Citado 2 vezes nas páginas 54 e 55.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software: Uma Abordagem profissional*. [S.l.]: New York: Higher Education, 9ª ed., 2016. Citado 6 vezes nas páginas 12, 40, 41, 42, 43 e 44.

RIBEIRO, L. *DevMedia*. 2012. Acessado em 16 de agosto de 2024. Disponível em: <https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408>. Citado na página 56.

RIVEST, R.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, fevereiro 1978. Citado na página 22.

STALLINGS, W. *Criptografia e Segurança de Redes: Princípios e Práticas*. 6. ed. São Paulo: Pearson Education do Brasi, 2015. Citado 4 vezes nas páginas 21, 23, 24 e 26.

STARLINE. 2024. Acessado em 16 de agosto de 2024. Disponível em: <https://br.freepik.com/vetores-gratis/moedas-de-ouro-bitcoin-criptomoeda-isoladas-em-branco_13891863.htm#fromView=search&page=1&position=2&uuid=989c6c0c-5d7a-45ce-9602-c2e7c12107b5>. Citado na página 34.

STELLA, J. C. Moedas virtuais no brasil: como enquadrar as criptomoedas. *Revista da PGBC*, v. 11, n. 2, p. 151, 2017. Citado na página 34.

SZABO, N. *Smart Contracts: Building Blocks for Digital Markets*. 1996. Acessado em 16 de agosto de 2024. Disponível em: <https://www.truevaluemetrics.org/DBpdfs/BlockChain/Nick-Szabo-Smart-Contracts-Building-Blocks-for-Digital-Markets-1996-14591.pdf>. Citado 2 vezes nas páginas 12 e 36.

VAZQUEZ, C. E.; SIMÕES, G. S.; ALBERT, R. M. *Análise de Pontos de Função: Medição, Estimativas e Gerenciamento de Projetos de Software*. São Paulo: Editora Érica, 2018. Citado 6 vezes nas páginas 45, 46, 47, 48, 50 e 52.

WERBACH, K.; CORNELL, N. Contracts ex machina. *Duke Law Journal*, v. 67, 2017. Citado 2 vezes nas páginas 12 e 37.