Geração Automática de Relatórios de Confiabilidade: Projeto e Implementação da Camada Front-end para Identificação de Causas de Falhas de Software na Plataforma X-RAT

Nathan Soares Mota

Universidade Federal de Uberlândia
Faculdade de Computação
Bacharelado em Sistemas de Informação

Monte Carmelo
2025

Nathan Soares Mota

Geração Automática de Relatórios de Confiabilidade: Projeto e Implementação da Camada Front-end para Identificação de Causas de Falhas de Software na Plataforma X-RAT

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção do grau Bacharel em Sistemas de Informação.

Área de concentração: Sistemas de Informação

Orientador: Caio Augusto Rodrigues dos Santos

Monte Carmelo
2025

Este trabalho é dedicado à minha família, especialmente aos meus pais, exemplos de força, apoio e dedicação ao longo de toda a minha formação.
É também dedicado a mim, pela resiliência diante dos desafios, pelas renúncias e pela determinação de persistir.
Dedico ainda este trabalho aos amigos que fiz nessa jornada, que tornaram o caminho mais leve e significativo.

Agradecimentos

Agradeço primeiramente a Deus, pela oportunidade e força ao longo desta jornada. Aos meus pais e à minha família, sou profundamente grato pelo apoio incondicional, tanto no aspecto emocional quanto no financeiro, por serem minha base nos momentos difíceis e minha fonte constante de motivação e inspiração. Sem o suporte de vocês, a realização deste trabalho não teria sido possível.

Agradeço também a mim mesmo, pela resiliência e determinação que me trouxeram até aqui. Por ter encarado a mudança para outro estado em busca da minha formação, por ter permanecido firme mesmo distante da família e dos amigos, e por ter enfrentado o período da pandemia de COVID-19, no qual precisei trabalhar e lidar com atrasos e desafios que estenderam minha trajetória acadêmica. Pelas noites em claro, pelos momentos de cansaço e pela persistência em continuar, apesar de todas as dificuldades — meu reconhecimento e gratidão a mim mesmo por não ter desistido.

Agradeço igualmente a todos os amigos que fiz nessa caminhada, em especial ao meu amigo Álvaro, parceiro essencial ao longo deste projeto. Compartilhamos inúmeros dias de desenvolvimento do sistema, reuniões e alinhamentos constantes por mais de um ano. Sua dedicação no backend, em conjunto com meu trabalho no frontend, foi fundamental para tornar este projeto possível.

"Eu sempre fui um sonhador, é isso que me mantém vivo!"
(Racionais MC's)

Resumo

Sistemas operacionais, como Windows 10 e 11, sustentam atividades críticas em ambientes pessoais e corporativos, de modo que falhas neste *software* podem comprometer a disponibilidade dos serviços e a experiência dos usuários. Nesse cenário, torna-se relevante analisar sistematicamente eventos de falha e estruturar informações que apoiem o diagnóstico de causas e o aprimoramento da confiabilidade dos computadores avaliados. Este trabalho apresenta o desenvolvimento e aprimoramento da plataforma X-RAT (*X-Reliability Analysis Tool*), uma ferramenta voltada para a análise e geração automática de relatórios de confiabilidade de *software* em sistemas operacionais (SOs), com foco nos Windows 10 e 11. A plataforma coleta *logs*, extrai eventos de falha, armazena os dados em um banco de dados e os processa por meio de algoritmos de categorização das falhas baseada em critérios como falhas de *kernel* ($\mathrm{OS_{KNL}}$), serviços ($\mathrm{OS_{SVC}}$), aplicações do sistema ($\mathrm{OS_{APP}}$) e aplicações do usuário ($\mathrm{USR_{APP}}$). Além disso, são calculadas diversas métricas estatísticas, incluindo a frequência das causas de falhas, a distribuição das falhas por categoria e a ocorrência dessas categorias ao longo do dia (Madrugada, Manhã, Tarde e Noite) e dos dias da semana. Por fim, a partir dos tempos entre falhas, aplicam-se métricas de confiabilidade, como MTBF (Mean Time Between Failures) e MTTF (Mean Time To Failure), bem como análises baseadas em distribuições de probabilidade. Cada análise é formalizada em um relatório gerado automaticamente pela plataforma. O objetivo deste trabalho é refatorar integralmente a plataforma previamente desenvolvida, que atualmente se encontra desatualizada e incompleta. A nova versão terá como diferencial a inclusão de funcionalidades adicionais, entre elas a coleta automática das falhas e diagnósticos mais precisos das causas dessas falhas, permitindo uma análise aprofundada e contribuindo para uma tomada de decisão mais informada no aprimoramento da confiabilidade dos computadores analisados.

Palavras-chave: Confiabilidade de *Software*. Sistemas Operacionais. Relatório de Confiabilidade. Causas de Falhas.

Abstract

Operating systems such as Windows 10 and 11 support critical activities in both personal and corporate environments, so failures in this *software* can compromise service availability and user experience. In this context, it becomes relevant to systematically analyze failure events and structure information that supports the diagnosis of root causes and the improvement of the reliability of the evaluated computers. This work presents the development and enhancement of the X-RAT platform (*X-Reliability Analysis Tool*), a tool designed for the analysis and automatic generation of software reliability reports in operating systems (OSs), focusing on Windows 10 and 11. The platform collects *logs*, extracts failure events, stores the data in a database, and processes them through failure categorization algorithms based on criteria such as kernel failures (OS_{KNL}), service failures (OS_{SVC}), system application failures (OS_{APP}), and user application failures (USR_{APP}). In addition, several statistical metrics are computed, including the frequency of failure causes, the distribution of failures by category, and the occurrence of these categories throughout the day (Early Morning, Morning, Afternoon, and Evening) and the days of the week. Finally, based on the times between failures, reliability metrics such as MTBF (Mean Time Between Failures) and MTTF (Mean Time To Failure) are applied, as well as analyses based on probability distributions. Each analysis is formalized in a report automatically generated by the platform. The goal of this work is to fully refactor the previously developed platform, which is currently outdated and incomplete. The new version will feature additional functionalities, including automatic failure collection and more accurate diagnostics of failure causes, enabling deeper analysis and contributing to more informed decision-making in improving the reliability of the analyzed computers.

Keywords: Reliability of Software. Operating Systems. Reliability Report. Failure Causes.

Lista de ilustrações

Lista de siglas

S0: Sistema Operacional (Operating System)

OSKNL: Falhas no Kernel (Kernel Failures)

OSSVC: Falhas em Serviços do Sistema Operacional (Operating System Services Failures)

OSAPP: Falhas em Aplicações do Sistema Operacional (Operating System Applications Failures)

USRAPP: Falhas de Aplicativos do Usuário (User Application Failures)

LGPD: Lei Geral de Proteção de Dados (General Data Protection Law)

MTBF: Mean Time Between Failures (Tempo Médio Entre Falhas)

MTTF: Mean Time To Failure (Tempo Médio Para Falha)

MTTR: Mean Time To Repair (Tempo Médio Para Reparo)

PDF: Probability Density Function (Função Densidade de Probabilidade)

X-RAT: Plataforma de Análise e Relatórios de Confiabilidade

API: Application Programming Interface (Interface de Programação de Aplicações)

Uptime Institute: Instituto de Estudos e Análise de Data Centers

ISO: International Organization for Standardization (Organização Internacional de Padronização)

IEC: International Electrotechnical Commission (Comissão Eletrotécnica Internacional)

Sumário

Capítulo 1

Introdução

Os Sistemas computacionais tornaram-se onipresentes na sociedade moderna, sendo amplamente utilizados em diversos setores essenciais, como saúde, comunicação, educação, negócios e serviços públicos (PETRICEK, 2020; BENNETT; BRACHMAN, 2010; ESPOSITO; MASTROIANNI, 2002; ABBAS; KHAN; ALI, 2023). Essa onipresença faz com que as falhas desses sistemas tenham um impacto significativo na vida humana, causando desde simples inconvenientes até danos catastróficos (MARTIN, 2023; SMITH, 2023). Dentre todas as ameaças ao correto funcionamento de sistemas computacionais, é amplamente conhecido que as falhas de *software* são predominantes (SALFNER; LUTZ; MALEK, 2006; MARTINO, 2014). De acordo com (INSTITUTE, 2021), falhas de *software* foram a principal causa de 119 interrupções em servidores relatadas publicamente em 2020.

Nesse contexto, a confiabilidade de *software* emerge como atributo crítico da engenharia de sistemas, referindo-se a capacidade de um *software* desempenhar corretamente suas funções sob condições específicas durante um período determinado (ANSI/IEEE, 1991). Esse conceito é fundamental para sistemas em que falhas podem resultar em sérias consequências, como a interrupção de serviços, perda de dados ou, em casos mais graves, riscos à segurança. A confiabilidade é frequentemente vista como uma métrica da qualidade do *software*, especialmente em ambientes críticos. Conforme estabelecido por normas internacionais, como a ISO/IEC 9126(AL-KILIDAR; COX; KITCHENHAM, 2005), confiabilidade pode ser definida pela capacidade do sistema de se recuperar de falhas ou erros, mantendo o comportamento dentro de limites aceitáveis. Garantir essa confiabilidade torna-se, portanto, essencial para prevenir interrupções e assegurar a continuidade dos serviços. Compreender como as falhas de *software* ocorrem é uma área de interesse não só para a academia, mas também para a indústria, pois pode levar a avanços significativos na confiabilidade de sistemas de *software*.

A análise sistemática de falhas em sistemas operacionais representa um desafio particularmente relevante, considerando que a maioria das aplicações depende desses sistemas para funcionar. Quando o sistema operacional falha, todas as aplicações que

operam sobre ele são potencialmente comprometidas. Os trabalhos de (PEIXOTO, 2023) e (LOPES, 2023) alcançaram avanços iniciais relevantes ao desenvolverem a primeira versão da plataforma X-RAT, voltada à análise de *logs* de falhas nos sistemas operacionais Windows 10 e 11. Esses estudos evidenciaram que a análise automatizada de *logs* possibilita a identificação de padrões recorrentes de falhas. No entanto, ambos se limitaram ao desenvolvimento do *Back-end* da plataforma.

Este trabalho surge, então, da necessidade de superar essas limitações através do desenvolvimento *Full Stack* da plataforma X-RAT, contemplando tanto o *Front-end* quanto uma refatoração completa do sistema. A proposta inclui a incorporação de funcionalidades avançadas, com destaque para a identificação automatizada das causas de falhas. O objetivo central é caracterizar as causas de falhas em computadores com Windows 10 e 11, identificando fatores contextuais que influenciam seu comportamento em diferentes ambientes de uso.

Além disso, a plataforma X-RAT é organizada em dois fluxos distintos e complementares: o fluxo administrativo e o fluxo do participante (usuário). No fluxo administrativo, acessível mediante autenticação, administradores gerenciam usuários e permissões, definem e mantêm os critérios de categorização, cadastram e editam as causas associadas aos códigos hexadecimais e mantêm a lista de eventos que não serão considerados falhas, atuando sobre um conjunto de *CRUD* voltados à curadoria e à configuração da ferramenta. No fluxo do participante, o usuário final interage com a interface *Front-end* para localizar e enviar os arquivos de log do Windows (Application.evtx e System.evtx), preencher o formulário de caracterização do ambiente (laboratorial, corporativo ou pessoal) e submeter os registros para processamento no *Back-end*. No processamento, os arquivos .evtx são extraídos, *timestamps* são normalizados com identificação do fuso real, eventos não-falha são filtrados, e os eventos são automaticamente classificados nas categorias *Kernel* do Sistema Operacional ($\mathrm{OS_{KNL}}$), Serviços do Sistema Operacional ($\mathrm{OS_{SVC}}$), Aplicativos do Sistema ($\mathrm{OS_{APP}}$) e Aplicativos do Usuário ($\mathrm{USR_{APP}}$). A partir daí, calcula-se a estatística das causas, distribuições temporais (períodos do dia e dias da semana), tempos entre falhas (TBF) e métricas de confiabilidade como MTBF e MTTF, além de testes de aderência para ajuste de distribuições probabilísticas; por fim, os resultados são apresentados ao usuário por meio de relatórios automáticos, tabelas e gráficos na interface internacionalizada da plataforma.

1.1 Motivação

A crescente dependência da sociedade moderna de sistemas computacionais torna a confiabilidade do *software* um aspecto essencial, especialmente em ambientes onde interrupções podem resultar em prejuízos operacionais, econômicos ou de segurança.

Embora os sistemas Windows 10 e 11 registrem detalhadamente eventos e falhas em seus arquivos de *log*, esses dados, por si só, não oferecem meios claros e acessíveis para que usuários ou analistas identifiquem padrões, compreendam as causas das falhas ou avaliem o nível de confiabilidade dos sistemas. Trabalhos anteriores relacionados à plataforma X-RAT demonstraram o potencial da análise automatizada desses registros, porém apresentavam limitações importantes, como ausência de coleta automática, falta de uma interface integrada e impossibilidade de identificar causas de falhas de forma sistemática.

Diante dessas lacunas, surge a necessidade de uma solução que automatize tanto a coleta quanto o processamento e a interpretação dos eventos registrados, apresentando-os de maneira clara e compreensível. A motivação deste trabalho está, portanto, na oportunidade de aprimorar a plataforma X-RAT para torná-la capaz de identificar causas de falhas, contextualizar sua ocorrência em diferentes ambientes e apresentar métricas de confiabilidade de forma integrada. Com isso, busca-se facilitar a análise de falhas em sistemas operacionais, tornando-a acessível não apenas a especialistas, mas também a profissionais e instituições que dependem da estabilidade de seus computadores para atividades cotidianas. A análise das causas de falhas em sistemas operacionais é crucial porque falhas podem originar-se em múltiplas camadas: no *kernel* (núcleo do sistema), em serviços do SO, em aplicações internas do sistema operacional ou em aplicações do usuário. Compreender essa pluralidade de origens é fundamental para uma avaliação abrangente da confiabilidade do sistema como um todo, evitando o viés de considerar apenas falhas no kernel ou em aplicações específicas.

1.2 Objetivos

1.2.1 Objetivo Geral

Investigar as causas de falhas em computadores que operam com os sistemas Windows 10 e 11, uma vez que são os sistemas operacionais *desktop* mais amplamente utilizados atualmente.

1.2.2 Objetivos Específicos

- ❏ Desenvolver a arquitetura *Full Stack* da plataforma X-RAT, contemplando tanto o *Front-end* quanto o *Back-end*;
- ❏ Implementar mecanismos de detecção automática de falhas a partir dos *logs* do sistema operacional;
- ❏ Criar sistema de categorização de falhas baseado nas classes: *kernel* ($\mathrm{OS_{KNL}}$), serviços ($\mathrm{OS_{SVC}}$), aplicações do sistema ($\mathrm{OS_{APP}}$) e aplicações do usuário ($\mathrm{USR_{APP}}$);

- ❏ Identificar as causas das falhas por meio da interpretação dos códigos hexadecimais com base na documentação oficial da Microsoft;
- ❏ Elaborar relatórios detalhados que ofereçam informações relevantes sobre o comportamento das falhas em ambientes de uso real;
- ❏ Implementar métricas de confiabilidade como MTBF (*Mean Time Between Failures*) e MTTF (*Mean Time To Failure*) para análise quantitativa.

1.3 Contribuições

Este trabalho contribui para o campo de confiabilidade de software ao:

1. Desenvolver uma nova versão da plataforma X-RAT com foco, mais especificamente, no *Front-end*, de modo a proporcionar uma visualização integrada e automatizada dos resultados de confiabilidade;
2. Implementar um método de identificação e categorização das causas de falhas de *software*;

1.4 Organização da Monografia

O presente trabalho está estruturado da seguinte forma:

No **Capítulo 2 - Fundamentação Teórica**, são apresentados os conceitos essenciais relacionados à confiabilidade de software e análise de falhas em sistemas operacionais. Esta seção aborda métricas fundamentais, como MTBF *(Mean Time Between Failures)* e MTTF *(Mean Time To Failure)*, além dos tipos de falhas mais recorrentes em sistemas Windows, categorizando-as em falhas de aplicação, serviços e *kernel*. Esses conceitos servem como base para a compreensão da análise de *logs* e a categorização das falhas, que são centrais para o desenvolvimento da plataforma X-RAT.

No **Capítulo 3 - Especificação da Plataforma**, é descrito o processo metodológico adotado para o desenvolvimento deste trabalho. São apresentadas as etapas de desenvolvimento e refatoração da plataforma X-RAT, detalhando as etapas de coleta, processamento e análise de *logs*. Este capítulo também aborda o ciclo de desenvolvimento e o estudo de caso que será conduzido para avaliar a eficácia da plataforma em ambientes variados, como laboratórios, empresas e uso pessoal.

No **Capítulo 4 - Resultados da Implementação** são apresentados os resultados da refatoração e implementação da plataforma X-RAT. Este capítulo descreve o fluxo completo da ferramenta, desde a coleta e envio dos arquivos de log do sistema operacional e o preenchimento do formulário de caracterização, até a análise e categorização das falhas. São exibidas as interfaces desenvolvidas, como o painel administrativo, os módulos

de cadastro e gerenciamento de causas de falhas, critérios de categorização e eventos não caracterizados, além da interface de análise e geração automática dos relatórios de confiabilidade. Também são mostrados os resultados visuais da plataforma, incluindo tabelas, gráficos e o suporte à internacionalização.

Capítulo 2

Fundamentação Teórica

2.1 Confiabilidade de *Software*

A confiabilidade de *software* é um dos atributos mais críticos na engenharia de sistemas, sendo responsável pela capacidade de um *software* desempenhar corretamente suas funções sob condições específicas durante um período determinado. Esse conceito é fundamental para sistemas em que falhas podem resultar em sérias consequências, como a interrupção de serviços, perda de dados ou, em casos mais graves, riscos à segurança. A confiabilidade é frequentemente vista como uma métrica da qualidade do *software*, especialmente em ambientes críticos. Conforme estabelecido por normas internacionais, como a ISO/IEC 9126(AL-KILIDAR; COX; KITCHENHAM, 2005), confiabilidade pode ser definida pela capacidade do sistema de se recuperar de falhas ou erros, mantendo o comportamento dentro de limites aceitáveis.

2.1.1 Métricas de Confiabilidade

Nesta seção, são apresentadas as principais métricas utilizadas para medir a confiabilidade de sistemas de *software*, conforme descrito no *Handbook of Software Reliability Engineering* (LYU et al., 1996). Cada métrica contribui para a avaliação do desempenho e da robustez do sistema ao longo do tempo, permitindo prever a ocorrência de falhas e planejar medidas de mitigação.

2.1.1.1 Confiabilidade

A confiabilidade de um sistema, denotada por $R(t)$, é a probabilidade de que o sistema funcionará corretamente durante um intervalo de tempo t, sem apresentar falhas. A fórmula é dada por:

$$R(t) = 1 - F(t)$$

Onde $F(t)$ é a função de distribuição de falhas, representando a probabilidade acumulada de falha até o tempo t.

2.1.1.2 *Warranty Time*

O *warranty time* é definido como o maior tempo entre falhas observado em um conjunto de dados, utilizado para determinar o tempo máximo de operação sem falhas. A fórmula para o *warranty time* é:

$$WT = \max(TBF)$$

Onde TBF é o tempo entre falhas observado no sistema.

2.1.1.3 Tempo Médio para Falha (MTTF)

O tempo médio para falha $MTTF$ é o valor esperado do tempo até que ocorra a primeira falha em um sistema não reparável. É calculado pela integral da função de confiabilidade:

$$MTTF = \int_0^\infty R(t)\,dt$$

2.1.1.4 Tempo Médio para Reparo (MTTR)

O tempo médio para reparo $MTTR$ é o tempo esperado para que um sistema seja restaurado após uma falha. A fórmula para $MTTR$ é dada por:

$$MTTR = \int_0^\infty t\,g(t)\,dt$$

Onde $g(t)$ é a função densidade de reparo, descrevendo o tempo necessário para restaurar o sistema.

2.1.1.5 Tempo Médio entre Falhas (MTBF)

O tempo médio entre falhas $MTBF$ é utilizado para sistemas que são reparáveis e representa o tempo médio de operação entre falhas. É dado pela soma do tempo médio para falha $MTTF$ e o tempo médio para reparo $MTTR$:

$$MTBF = MTTF + MTTR$$

2.2 Falhas de *Software*

Falhas de software são eventos indesejados que ocorrem quando o serviço fornecido por um sistema se desvia do serviço correto, ou seja, quando o sistema deixa de executar

a função para a qual foi projetado, entregando um resultado incorreto. Seguindo a taxonomia descrita por Avizienis et al. (AVIZIENIS et al., 2004) referenciada em (SANTOS; JR.; TRIVEDI, 2021), podemos distinguir três conceitos principais relacionados às falhas de software:

1. **Falha**: Um evento que ocorre quando o serviço fornecido pelo sistema se desvia do serviço correto, ou seja, quando o sistema não realiza exatamente o que se espera. Uma falha nem sempre resulta em uma interrupção completa (como um travamento), podendo ocorrer em formas mais sutis que afetam a funcionalidade esperada do software sem necessariamente causar um bloqueio ou encerramento do sistema.

2. **Erro**: Refere-se à consequência de uma falta, onde o estado incorreto gerado pela falta causa uma diferença entre o serviço esperado e o serviço entregue pelo sistema. Esse estado incorreto pode ser perceptível ou não, mas representa uma discrepância em relação ao funcionamento correto do sistema.

3. **Falta (Fault)**: A causa de um erro, geralmente relacionada a um defeito ou bug no código do software. A falta é a origem que, quando ativada sob determinadas condições, desencadeia uma sequência que leva ao erro e, eventualmente, à falha observada no serviço do sistema.

Essa definição permite considerar que falhas de *software* nem sempre causam interrupções completas, como travamentos ou encerramentos, mas podem incluir eventos que, embora não desativem o sistema, resultam em um serviço que se desvia do esperado. Essa abordagem permite uma análise mais abrangente das falhas e evita o viés de seleção ao considerar apenas falhas que resultam em travamentos ou interrupções completas do sistema (SANTOS; JR.; TRIVEDI, 2021).

2.3 Categorias de Falhas

A categorização de falhas de *software* pode ser dividida em dois principais níveis de análise: falhas do SO e falhas de aplicações do usuário. Essa categorização permite uma análise detalhada dos diferentes pontos onde as falhas podem ocorrer. Para o sistema operacional, as falhas podem ocorrer no núcleo do sistema, em serviços operacionais em segundo plano ou em aplicações que interagem diretamente com o usuário. A seguir, são descritas essas categorias:

1. **Falhas do SO**:

a) **Falhas no *Kernel* ($\mathbf{OS_{KNL}}$)**: Ocorrências de falhas no código do *kernel* do sistema operacional, que frequentemente resultam em travamentos ou reinicializações, devido ao impacto direto no gerenciamento de hardware e na execução de funções críticas do sistema.

b) **Falhas em Serviços do Sistema Operacional ($\mathbf{OS_{SVC}}$)**: Esses serviços operam no espaço de usuário, executando em segundo plano para fornecer funcionalidades como gerenciamento de impressão, agendamento de tarefas e eventos do sistema. Embora essas falhas nem sempre causem uma interrupção completa, elas resultam em uma degradação do serviço fornecido, afetando a confiabilidade percebida pelo usuário.

c) **Falhas em Aplicações do Sistema Operacional ($\mathbf{OS_{APP}}$)**: Compreende falhas que ocorrem em aplicações internas do sistema, como o gerenciador de janelas. Essas falhas impactam diretamente a experiência do usuário, mas sem afetar o núcleo do sistema.

2. **Falhas de Aplicações do Usuário ($\mathbf{USR_{APP}}$)**: Falhas que ocorrem nas aplicações de usuário, que são os programas com os quais o usuário interage diretamente, como editores de texto, navegadores e outros *softwares* de produtividade. Como essas falhas acontecem no espaço de usuário, elas não interferem diretamente no funcionamento do sistema operacional em si, mas comprometem a funcionalidade e a experiência do usuário ao utilizar a aplicação específica.

Essa categorização considera que falhas em qualquer uma dessas camadas podem desviar o serviço fornecido pelo sistema do serviço correto esperado pelo usuário, mesmo que não resulte necessariamente em um travamento ou reinicialização. Ao analisar as falhas em múltiplas camadas do sistema operacional, este modelo permite uma avaliação mais precisa da confiabilidade do sistema operacional como um todo, conforme recomendado em estudos como o de (SANTOS; JR.; TRIVEDI, 2021), que indicam a importância de não limitar a análise apenas a falhas no *kernel* para uma avaliação completa da confiabilidade do sistema.

2.4 Abordagem de desenvolvimento de *software*

Segundo (SOMMERVILLE, 2018), a seleção de uma abordagem de desenvolvimento de *software* pode ser orientada por uma série de perguntas-chave, que exploram aspectos técnicos, humanos e organizacionais do sistema, da equipe de desenvolvimento e dos *stakeholders* envolvidos. Esses fatores permitem uma análise aprofundada para escolher entre abordagens de desenvolvimento dirigidas por plano ou ágeis. A Figura 1 apresenta os fatores considerados para a escolha de cada abordagem.

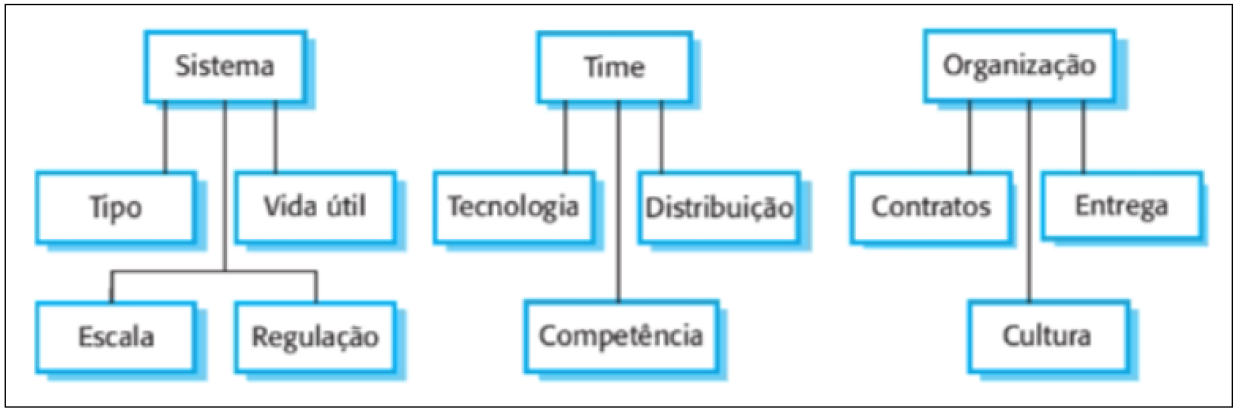


Figura 1 – Fatores que influenciam a escolha entre desenvolvimento dirigido por plano ou ágil. Fonte: Sommerville, 2018.

Questões Técnicas

1. **Qual é o tamanho do sistema que está sendo desenvolvido?** É um *software* de pequeno porte.

2. **Que tipo de sistema está sendo desenvolvido?** Uma aplicação *web* de complexidade de pequena a média.

3. **Qual é a vida útil prevista para o sistema?** A vida útil prevista para o sistema é classificada como de média duração, com base na vida útil média dos sistemas operacionais Windows 10 e 11, que é de aproximadamente 10 anos (Microsoft, 2023). Isso é conforme a política de suporte da Microsoft, que divide esse período em 5 anos de suporte mainstream e 5 anos de suporte estendido, com garantias de atualizações de segurança e correções críticas.

4. **O sistema está sujeito a controle externo?** Sim, o sistema está sujeito a controle externo, pois deve atender às regulamentações de segurança e privacidade estabelecidas pela LGPD no Brasil (BRASIL, 2018).

Questões Humanas

1. **Qual é o nível de competência dos projetistas e programadores do time de desenvolvimento?** O time é multidisciplinar, com pessoas tendo experiência e sendo competentes em áreas diversas do desenvolvimento de projetos de *software*.

2. **Como está organizado o time de desenvolvimento?** O time é pequeno, e apesar de terem papéis definidos, todos devem participar da maioria das atividades.

3. **Quais são as tecnologias disponíveis para apoiar o desenvolvimento do sistema?** Há um bom escopo ferramental e tecnológico.

Considerando as respostas obtidas com base nas perguntas propostas por (SOMMERVILLE, 2018), foi possível chegar às seguintes conclusões:

O *software* a ser desenvolvido é de porte pequeno, com complexidade baixa a média e vida útil curta, o que torna viável a aplicação de uma metodologia ágil, pois não exige um esforço intenso de documentação para comunicação entre a equipe e o cliente. A equipe de desenvolvimento é composta por membros experientes e familiarizados com as tecnologias necessárias, utilizando ferramentas produtivas que facilitam a colaboração e comunicação interna. Essas características reforçam a adequação de uma metodologia ágil, já que o trabalho em equipe ocorre de forma eficiente. A implementação pode ser iniciada antes que o produto esteja totalmente especificado, permitindo entregas incrementais e rápidas, que podem ser ajustadas com base em *feedbacks*. Portanto, a abordagem mais adequada para este projeto é o desenvolvimento de *software* ágil.

2.5 Trabalhos Relacionados

Esta seção apresenta e discute os trabalhos relacionados, destacando as contribuições relevantes no contexto deste estudo. A seguir, são descritos os principais projetos e pesquisas realizados, incluindo os trabalhos desenvolvidos pelo autor, que se relacionam com o tema abordado.

O trabalho de Oliveira (OLIVEIRA, 2018) foca na análise da confiabilidade do sistema operacional Windows 10, que utilizou a ferramenta OSRat para explorar dados de falhas de *software*. Este estudo contribui ao estruturar um processo de extração e organização dos dados dos arquivos de *log* do sistema, seguido por uma análise criteriosa que facilita a classificação das falhas em diferentes categorias. Essa abordagem é essencial para compreender os desafios de confiabilidade específicos do Windows 10, um dos sistemas mais utilizados atualmente, e serve como base para o aprimoramento de futuras pesquisas na plataforma X-RAT, que busca analisar e prever falhas de forma mais abrangente.

No trabalho de Santos et al. (SANTOS; MATIAS; TRIVEDI, 2020), foi proposto um método estatístico para prever falhas de sistemas operacionais com base na associação de múltiplos eventos de falha. Este estudo se destaca pela aplicação de técnicas avançadas de análise de dados, que permitiram identificar correlações entre falhas consecutivas. A abordagem desenvolvida possibilita uma análise mais profunda da confiabilidade do sistema, fornecendo insights para melhorar a prevenção de falhas e a manutenção de sistemas críticos.

Santos et al. (SANTOS; JR.; TRIVEDI, 2021) realizaram uma análise detalhada das causas de falhas tanto em sistemas operacionais quanto em *softwares* de aplicação em múltiplos locais. O estudo examinou falhas de diversos tipos, visando identificar padrões de falhas que impactam a confiabilidade de *softwares* amplamente utilizados, com foco na

variação entre diferentes ambientes operacionais. Esta pesquisa é relevante para entender como diferentes fatores ambientais influenciam as causas de falhas e para a implementação de abordagens preventivas.

O trabalho de Diogo Canut Freitas Peixoto (PEIXOTO, 2023) contribuiu com o desenvolvimento da plataforma X-RAT que implementou uma *engine* dedicada à análise automatizada de *logs* de falhas, focada em sistemas operacionais Windows. Esse projeto facilitou o processo de leitura e categorização de eventos de falha, estabelecendo uma base sólida para a geração automática de relatórios de confiabilidade. A partir dessa estrutura, o presente trabalho amplia o escopo da X-RAT, direcionando a análise para grupos de computadores em diferentes ambientes e explorando como as características desses contextos influenciam a frequência e o tipo de falhas. Assim, o estudo atual se apoia nas funcionalidades desenvolvidas por Peixoto, estendendo-as para uma aplicação que abrange ambientes variados e possibilita uma compreensão mais ampla dos fatores que impactam a confiabilidade de sistemas.

O trabalho de Bruno Coelho Lopes (LOPES, 2023) contribuiu com o aprimoramento da plataforma X-RAT, que introduziu métodos de análise preditiva baseados em padrões de eventos múltiplos, com foco na antecipação de falhas em sistemas operacionais Windows. Essa pesquisa forneceu uma base sólida de técnicas que permitiram identificar padrões de falhas recorrentes, estabelecendo uma estrutura inicial para a análise de confiabilidade. A partir desses avanços, o presente trabalho expande a análise da X-RAT para múltiplos grupos de computadores em diferentes ambientes, explorando como fatores contextuais influenciam a ocorrência de falhas. Dessa forma, a pesquisa atual se apoia nas melhorias propostas por Lopes, mas direciona o foco para a análise em contextos variados, ampliando a aplicabilidade e a precisão da plataforma em ambientes específicos.

2.6 Síntese e Comparação dos Trabalhos Relacionados

Os trabalhos analisados oferecem fundamentos conceituais e experimentais essenciais para o desenvolvimento deste projeto, especialmente no que se refere à caracterização de falhas de software, confiabilidade de sistemas operacionais e métodos de análise de eventos. Em particular, estudos como o de Matias e Oliveira, bem como as pesquisas de Santos, Matias e Trivedi, reforçam a necessidade de avaliar a confiabilidade do sistema operacional considerando múltiplas categorias de falhas, o que vai ao encontro da taxonomia empregada na plataforma X-RAT (SANTOS; JR.; TRIVEDI, 2021; JR. et al., 2013; JR. et al., 2014; OLIVEIRA, 2018). Esses trabalhos também evidenciam a relevância de analisar dados reais provenientes de ambientes diversos, o que fundamenta as escolhas metodológicas adotadas neste TCC. Além disso, pesquisas voltadas para a predição de falhas e associação múltipla de eventos destacam padrões recorrentes

presentes em ambientes reais, fornecendo subsídios importantes para a extensão e aprimoramento da X-RAT (SANTOS; MATIAS; TRIVEDI, 2020; LOPES, 2023). Esses estudos influenciam diretamente a abordagem de extração, processamento e análise estatística empregada, reforçando a importância de métricas como MTBF, distribuições de falhas e categorização detalhada dos eventos.

Por outro lado, os trabalhos existentes não contemplam elementos essenciais abordados neste projeto. Nenhum deles apresenta uma solução integrada com *pipeline* completo — desde a coleta automatizada dos arquivos de *log* até a geração dinâmica e estruturada de relatórios. Também não há, nessas pesquisas, a implementação de uma interface moderna, mecanismos de administração da plataforma ou a criação de um módulo dedicado exclusivamente à identificação estruturada das causas de falhas, de forma interligada ao fluxo computacional e ao banco de dados. A ausência de uma plataforma operacional completa limita o uso prático dos resultados apresentados na literatura.

Dessa forma, este TCC se posiciona como uma evolução prática em relação aos trabalhos relacionados, ao transformar conceitos e métodos previamente estudados em uma solução aplicada, automatizada e utilizável em ambientes reais. A nova versão da plataforma X-RAT, reconstruída e estendida neste trabalho, integra coleta, categorização, análise estatística e visualização, oferecendo uma ferramenta completa para diagnóstico e geração automática de relatórios de confiabilidade. Assim, o presente trabalho estabelece um elo entre a base teórica existente e uma aplicação concreta capaz de apoiar decisões técnicas relacionadas à confiabilidade de sistemas operacionais, complementando e ampliando as contribuições encontradas na literatura (PEIXOTO, 2023).

2.7 Comparação com versões anteriores

A versão inicial da plataforma X-RAT apresentava diversas limitações funcionais que restringiam sua capacidade de apoiar análises de confiabilidade de *software*. Embora representasse um avanço importante no sentido de centralizar informações provenientes de *logs* do sistema operacional, essa versão ainda dependia de procedimentos manuais, ferramentas externas e arquivos auxiliares para realizar tarefas fundamentais.

Entre as principais limitações observavam-se: a ausência de um mecanismo automatizado de coleta de eventos; a dependência de arquivos CSV externos para filtragem e categorização de falhas; a falta de autenticação de usuários; restrições no tratamento adequado de *timestamps*; e a inexistência de informações relacionadas às causas das falhas registradas. Essas limitações motivaram a evolução da plataforma, cujo aprimoramento é apresentado posteriormente na seção de Resultados Alcançados.

A Tabela 1 resume as principais características da primeira versão da X-RAT, destacando os aspectos que precisavam ser ampliados ou reestruturados.

Funcionalidade	**Primeira Versão do X-RAT**
Coleta de *logs* de falhas	Não disponível – os *logs* precisavam ser coletados manualmente pelo usuário.
Mecanismo de filtro de eventos	Dependente de arquivo CSV externo carregado manualmente pela aplicação.
Mecanismo de categorização de falhas	Dependência de arquivo CSV externo contendo categorias pré-definidas.
Autenticação (Login)	Não possuía sistema de *login*; o acesso era totalmente aberto.
Transformação de *timestamp*	Conversão automática fixa para o fuso horário brasileiro (GMT-3), sem considerar o fuso real da máquina onde ocorreu a falha.
Formato do relatório	Geração de arquivo PDF estático após o processamento dos eventos.
Informações sobre causas das falhas	Não disponível – apenas o registro do evento era apresentado, sem interpretar o código associado.

Tabela 1 – Funcionalidades presentes na primeira versão da plataforma X-RAT.

Capítulo 3

Especificação da Plataforma

3.1 Visão Geral

Este capítulo descreve o método utilizada para atingir os objetivos da pesquisa, focando na análise de falhas de *software* em grupos de computadores e na geração automática de relatórios de confiabilidade na plataforma X-RAT. A abordagem segue uma sequência estruturada de passos, desde a coleta de *logs* e a categorização de eventos até a análise estatística e a geração de relatórios com métricas de confiabilidade. O desenvolvimento técnico foi realizado utilizando Python[1] com FastAPI[2] para o *Back-end*, PostgreSQL[3] como banco de dados e React[4] com Vite[5] e TypeScript[6] para o *Front-end*.

3.2 Dados de Falha

O sistema operacional Windows registra automaticamente *logs* detalhados de falhas, abrangendo tanto os componentes do sistema quanto as aplicações do usuário. Esses registros são essenciais para monitoramento e análise de falhas, permitindo que informações relevantes sobre o comportamento do sistema sejam armazenadas e recuperadas para diagnóstico e solução de problemas.

Os *logs* de falhas no Windows 10 e 11 estão localizados em um diretório específico, geralmente acessível em `C:\Windows\System32\winevt\Logs`. Neste local, os arquivos de *log* de eventos são salvos no formato `.evtx`, um padrão de armazenamento estruturado que permite a organização dos dados de eventos e facilita a recuperação de informações para análise.

[1] Disponível em: <https://www.python.org/>
[2] Disponível em: <https://fastapi.tiangolo.com/>
[3] Disponível em: <https://www.postgresql.org/>
[4] Disponível em: <https://react.dev/>
[5] Disponível em: <https://vitejs.dev/>
[6] Disponível em: <https://www.typescriptlang.org/>

O formato `.evtx` foi projetado para suportar um grande volume de dados e fornecer uma estrutura otimizada para a captura de eventos de sistema, incluindo detalhes como o tipo de evento, a origem, a data e a hora da ocorrência, entre outros atributos essenciais para a análise (METZ, 2023). Essa estrutura permite que ferramentas de análise, como a plataforma X-RAT, acessem e processem os dados de falhas, possibilitando a identificação de padrões de comportamento e causas recorrentes de falhas no sistema.

3.3 Método

Para analisar e processar os dados de falha, é necessário seguir uma série de etapas que permitem a extração, categorização e análise dos eventos registrados nos arquivos de *log* do Windows. A Figura 4 ilustra o fluxo completo do sistema, que começa com a coleta de *logs* de falhas e informações de caracterização no *Front-end*, passa por diversas etapas de processamento e análise no *Back-end* e, por fim, gera relatórios detalhados.

3.3.1 Pré-processamento

A Figura 2 apresenta a primeira parte do fluxo da plataforma X-RAT que corresponde ao pré-processamento, que envolve autenticação, definição de critérios de categorização e filtragem dos eventos não caracterizados como falhas.

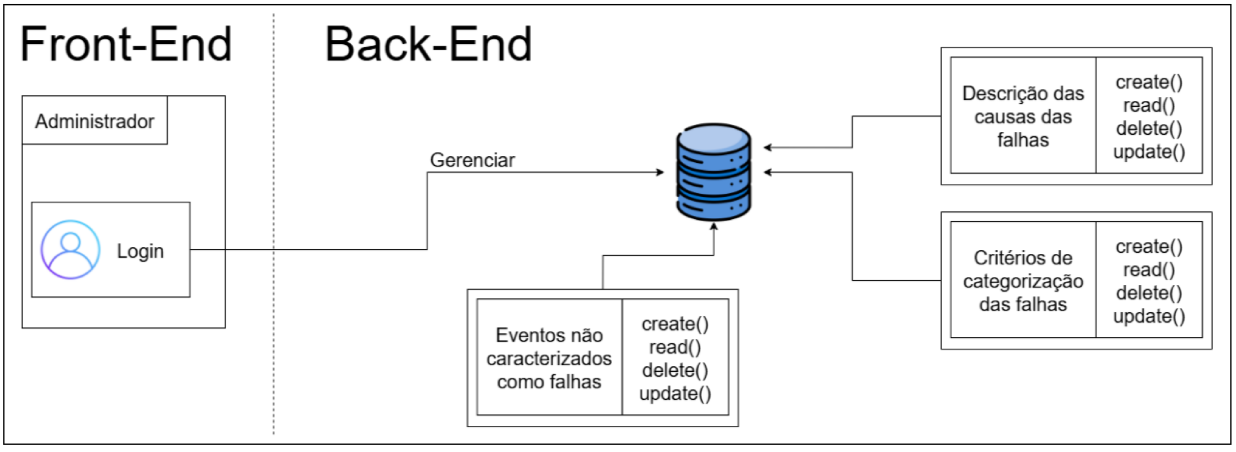


Figura 2 – Primeira parte do fluxo da plataforma X-RAT

- ***Login***.

  Foi criado um *login* de administrador para o sistema, permitindo a realização de operações de *CRUD* nos três itens descritos abaixo. Esse administrador deve ser um especialista em eventos de falha de sistemas operacionais.

- **Critérios de categorização das falhas**.

Com base nos diferentes campos dos eventos de falha, é possível categorizá-los inicialmente em falhas do sistema operacional e falhas de aplicações de usuário. No caso do sistema operacional, as falhas podem ser classificadas de forma mais detalhada em três grupos: falhas no núcleo do sistema, em serviços operacionais de segundo plano ou em aplicações que interagem diretamente com o usuário conforme apresentado na Seção 2.3.

- **Descrição das causas das falhas**.

A análise das causas das falhas é conduzida a partir da identificação dos códigos hexadecimais associados aos eventos, os quais são posteriormente interpretados com base na documentação oficial da Microsoft (Microsoft, 2021).

- **Eventos não caracterizados como falhas**.

Existem situações em que determinados eventos não devem ser considerados falhas. Por exemplo, o evento com ID 4319 referente ao NetBT, apenas indica que um nome duplicado foi detectado na rede TCP (MICROSOFT, 2021). Portanto, este evento não pode ser considerado uma falha. Desta forma, foi criado este módulo em que são definidos os eventos que não devem ser classificados como falhas. Durante o processo de extração dos registros a partir dos arquivos *.evtx*, caso um evento pertença a essa lista, ele é automaticamente desconsiderado, não sendo incluído na base de dados do sistema.

.

3.3.2 Processamento dos *Logs* de Falha

A segunda parte do fluxo é apresentada na Figura 3, englobando as etapas de coleta dos *logs*, extração de eventos de falha, caracterização e geração de relatórios automáticos.

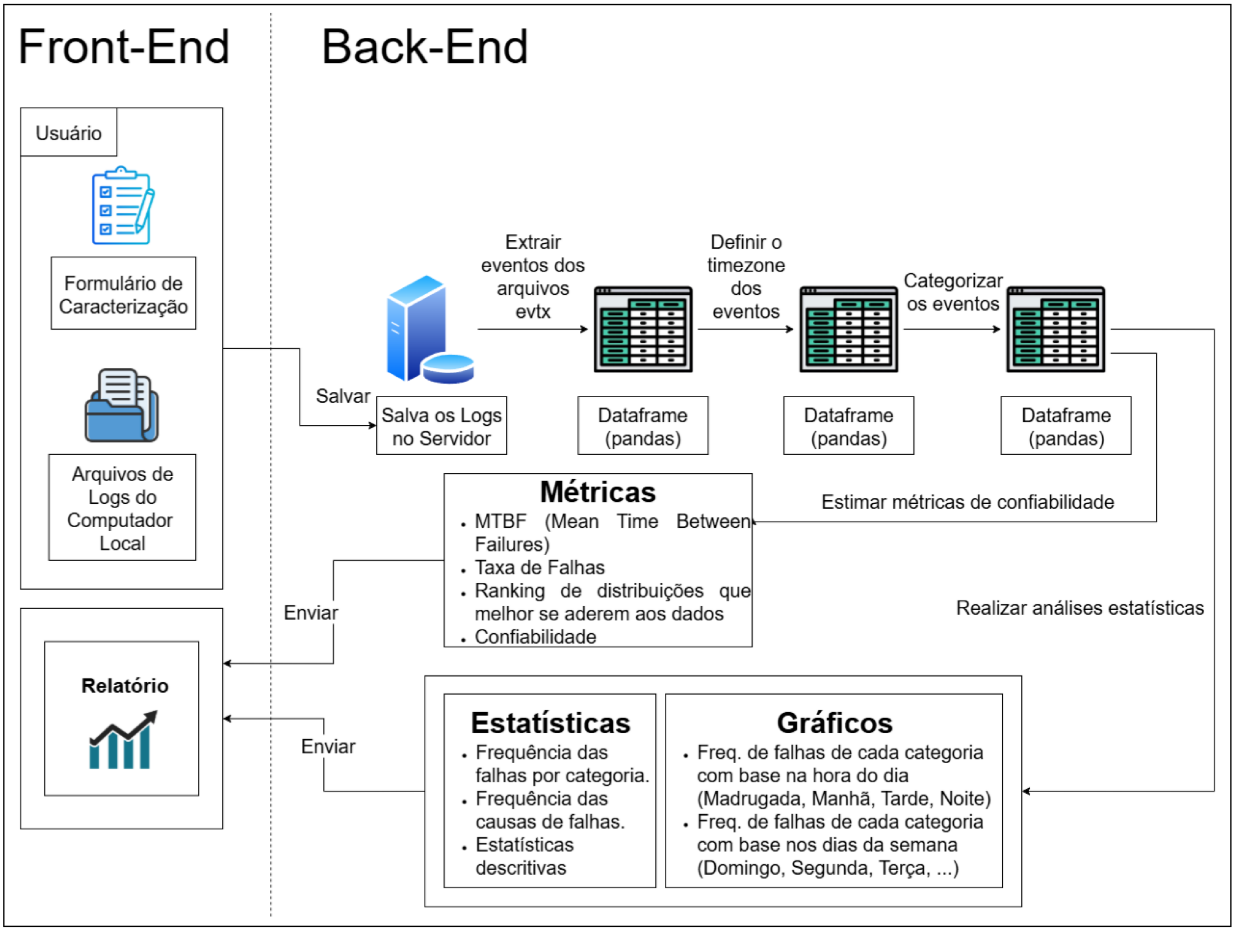


Figura 3 – Segunda parte do fluxo da plataforma X-RAT

❑ **Coleta dos *logs*.**

Nessa etapa, os *logs* de falha do Windows são coletados por meio do *upload* no *website*, que fornece instruções passo a passo para a execução do processo e, em seguida, direciona o usuário ao preenchimento de um formulário de caracterização do uso do sistema. O formulário captura informações essenciais sobre o contexto de cada dispositivo como tipo de ambiente (laboratorial, empresarial, pessoal) e características de uso do sistema. Essas informações são utilizadas posteriormente para contextualizar a análise de falhas em diferentes cenários e responder a perguntas específicas de pesquisa sobre padrões e causas de falhas em diferentes grupos.

Os arquivos de *log* enviados pelo usuário são salvos no servidor em diretórios únicos, enquanto o caminho desses arquivos e o formulário são persistidos no banco de dados para posterior processamento.

❑ **Extração dos eventos de falha.**

Os arquivos de *log* são enviados ao *Back-end* da plataforma X-RAT, onde são processados e os eventos de falha são extraídos. Durante a extração, atributos

como *timestamp* e mensagem são registrados, permitindo uma manipulação estruturada dos dados e facilitando as etapas subsequentes de análise. Além disso, eventos identificados como não falhas são filtrados automaticamente. Esses registros são armazenados em um *CRUD* administrativo, que possibilita consultar, filtrar, editar e remover elementos da lista conforme apresentado na Seção 3.3.1

- **Definição do *Timezone*.**

  Para garantir a consistência dos dados extraídos, todos os eventos de falha registrados no sistema são inicialmente configurados em um fuso horário padrão. Essa padronização é essencial, uma vez que os eventos mantêm o mesmo fuso horário independentemente da localização geográfica do dispositivo de origem. No entanto, é importante identificar o momento exato em que as falhas ocorreram. Assim, a partir dos campos de data e hora presentes nos registros do *Reliability Analysis Component (RAC)*, foi possível determinar o fuso horário real da ocorrência de cada falha, permitindo ajustar o campo temporal de cada evento ao horário local correspondente.

- **Categorização das Falhas e Identificação de suas Causas**.

  As falhas são classificadas em diferentes grupos, conforme a categoria identificada. Utilizando a taxonomia descrita, os eventos são classificados nas seguintes categorias principais: ***Kernel* do Sistema Operacional ($\mathbf{OS_{KNL}}$)**, **Serviços do Sistema Operacional ($\mathbf{OS_{SVC}}$)**, **Aplicativos do Sistema Operacional ($\mathbf{OS_{APP}}$)** e **Aplicativos do Usuário ($\mathbf{USR_{APP}}$)**. Essa classificação é fundamental para o estudo, pois permite uma análise detalhada das falhas conforme sua origem e oferece uma visão estruturada dos diferentes tipos de eventos que afetam a confiabilidade.

  Os códigos que indicam as causas das falhas são atribuídos automaticamente pelo sistema operacional no momento da ocorrência. Portanto, assume-se que estejam corretos. Esses códigos, representados em formato hexadecimal, identificam de forma única as causas das falhas. A interpretação dos valores é realizada com base na documentação oficial do sistema operacional, o que pode estar sujeito a erros de interpretação. Entretanto, a maioria desses códigos possui descrições diretas, representando uma ameaça mínima à validade deste estudo.

- **Relatório com Métricas, Estatísticas e Gráficos**.

  Para calcular as métricas apresentadas na Seção 2.1.1, é necessário, primeiramente, aplicar testes de aderência aos tempos entre falhas (*Time Between Failures* – TBFs), a fim de identificar a distribuição de probabilidade que melhor se ajusta aos dados. Além disso, foram gerados gráficos e estatísticas para analisar os horários e os dias da semana em que as falhas ocorrem com maior frequência.

Os relatórios finais incluem toda a análise realizada e são apresentados ao usuário no *Front-end* da plataforma.

Após a apresentação separada das duas etapas principais — o pré-processamento e o processamento dos *logs* de falha — torna-se possível visualizar o funcionamento integrado da plataforma. Dessa forma, a Figura 4 apresenta o fluxo completo do sistema, reunindo em um único diagrama todas as fases descritas anteriormente. Essa visão consolidada permite compreender como cada componente interage e como as etapas individuais se articulam para compor o processo de análise automatizada de confiabilidade.

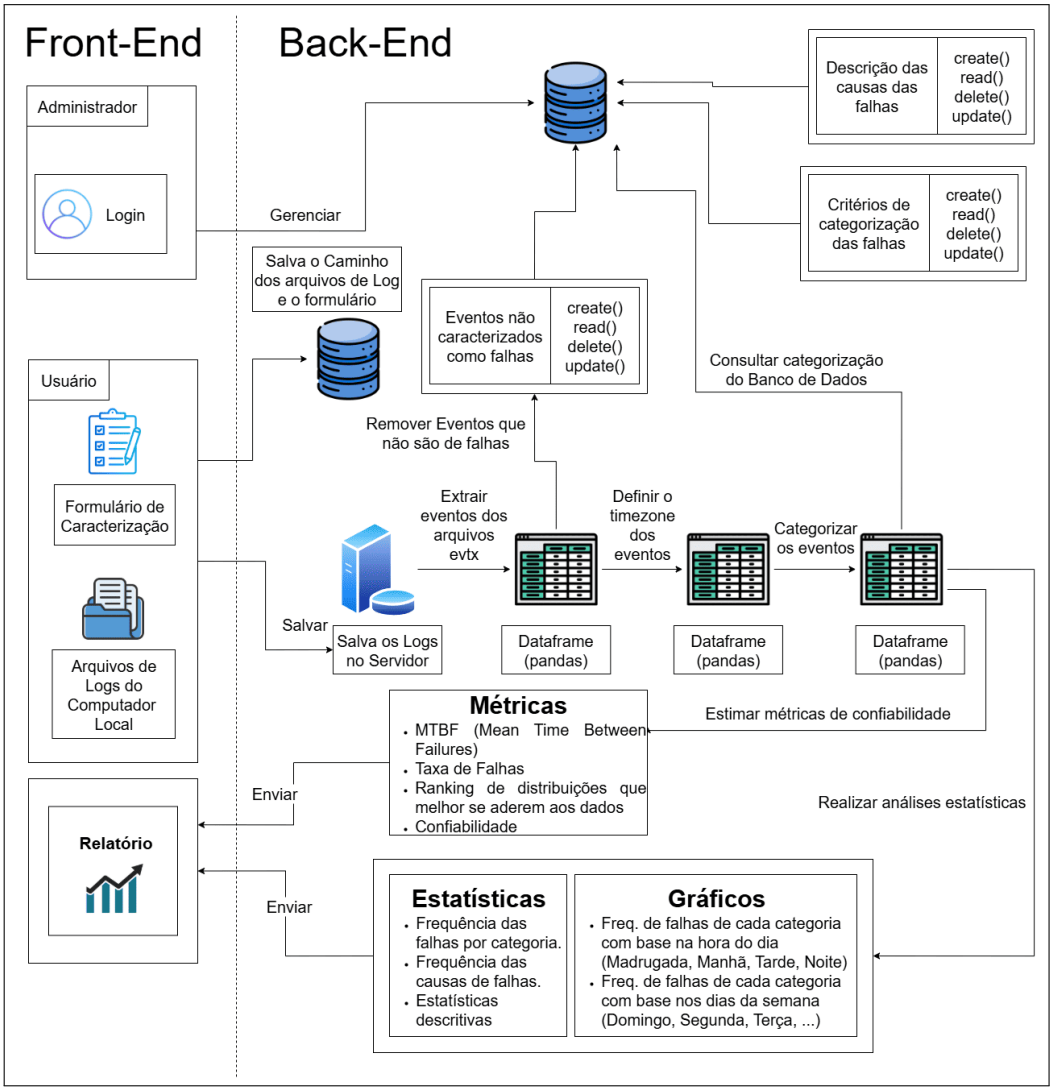


Figura 4 – Fluxo geral da plataforma X-RAT

3.4 Tecnologias Utilizadas no Front-end

Esta seção descreve as tecnologias que foram empregadas no desenvolvimento do *Front-end* da plataforma X-RAT, cuja interface foi construída com *React* e *TypeScript*. A escolha

das ferramentas teve como base critérios de desempenho, manutenibilidade, suporte a padrões modernos de desenvolvimento e compatibilidade com os requisitos funcionais e não funcionais do sistema.

3.4.1 Vite

A escolha do *Vite* como ferramenta de *build* e desenvolvimento para o *Front-end* da plataforma X-RAT foi motivada por sua arquitetura moderna e desempenho superior em comparação ao *Create React App* tradicional. Segundo Gurung (GURUNG, 2022), o Vite se destaca por adotar uma abordagem baseada em módulos ES nativos para desenvolvimento, o que resulta em tempos de inicialização extremamente rápidos e recarregamento instantâneo de módulos alterados, sem a necessidade de reconstruções completas. Além disso, a estratégia de pré-compilação do Vite, aliada ao uso do *esbuild*, um empacotador escrito em *Go*, proporciona uma performance significativamente melhor tanto em ambientes de desenvolvimento quanto em *builds* de produção. Tais vantagens são especialmente relevantes para este projeto, que demanda agilidade nas iterações e tempo de resposta eficiente no *Front-end*.

3.4.2 Internacionalização

A internacionalização (*i18n*) da interface da plataforma X-RAT foi implementada utilizando a biblioteca *i18next*, em conjunto com o módulo *react-i18next*, permitindo a tradução dinâmica de todos os textos exibidos no *Front-end*. Essa abordagem viabiliza que a aplicação seja apresentada em diferentes idiomas — atualmente português e inglês — de forma totalmente automatizada e sem a necessidade de recarregar a página.

A configuração do sistema de tradução é realizada por meio de dois arquivos principais, `pt.json` e `en.json`, que armazenam, em formato *key-value*, todos os textos da interface. Cada chave representa um identificador semântico do texto, enquanto o valor associado contém a tradução correspondente no idioma desejado. Dessa forma, a troca de idioma ocorre de maneira instantânea, bastando alterar o parâmetro linguístico ativo.

Para detectar automaticamente o idioma preferencial do usuário, a aplicação utiliza o módulo *i18next-browser-languagedetector*, que identifica o idioma configurado no navegador e o aplica como padrão na inicialização. Caso o idioma detectado não esteja disponível, a aplicação recorre ao idioma de *fallback*, definido como inglês.

O processo de inicialização do *i18n* ocorre no arquivo principal da aplicação (*main.tsx*), onde o módulo é importado e integrado ao *React*. Essa arquitetura garante flexibilidade na adição de novos idiomas e mantém a coerência textual em todos os componentes da interface. Com isso, a plataforma X-RAT torna-se acessível a um público mais amplo e adequada a contextos acadêmicos e corporativos internacionais.

3.4.3 Tema Escuro (*Dark Mode*)

A plataforma X-RAT oferece suporte à alternância entre os modos claro e escuro, com o objetivo de melhorar a experiência do usuário em diferentes condições de luminosidade e preferências visuais. Essa funcionalidade foi implementada utilizando o *Tailwind CSS* em conjunto com um provedor de contexto personalizado denominado *ThemeProvider*.

O *ThemeProvider* é responsável por gerenciar o estado global do tema e armazenar a preferência do usuário no *localStorage*, garantindo que a configuração escolhida permaneça ativa entre sessões de uso. A definição inicial do tema segue a configuração de cor do sistema operacional do usuário, detectada por meio da propriedade *prefers-color-scheme*. Internamente, essa propriedade utiliza a API de *Media Queries* do JavaScript, acessada pelo método *window.matchMedia()*, que identifica o modo de cor (claro ou escuro) configurado no navegador do usuário e ajusta a interface automaticamente conforme essa preferência.

A alternância entre os temas é realizada dinamicamente pela adição ou remoção da classe *dark* no elemento raiz do documento. O *Tailwind CSS* simplifica a aplicação de estilos condicionais, permitindo que os componentes da interface utilizem o prefixo *dark:* para definir variações de cor, sombra e contraste. Dessa forma, um mesmo componente pode apresentar automaticamente aparências diferentes conforme o tema ativo, sem necessidade de duplicar folhas de estilo.

Essa abordagem garante uma experiência visual consistente, reduz o cansaço ocular em ambientes com pouca iluminação e reforça o foco da plataforma em acessibilidade e personalização da interface. O suporte ao modo escuro, aliado à internacionalização, contribui para tornar o *Front-end* da X-RAT mais inclusivo, moderno e adaptável às preferências dos usuários.

3.4.4 Roteamento

A navegação entre as diferentes páginas da aplicação foi implementada com a biblioteca *React Router DOM*, que fornece mecanismos para controle de rotas e estruturação lógica de componentes e layouts. Esse recurso permitiu a organização clara entre áreas públicas e privadas da aplicação, além de facilitar a manutenção e escalabilidade do sistema.

3.4.5 Gerenciamento de Estado e Cache

O gerenciamento de dados assíncronos foi realizado com a biblioteca *@tanstack/react-query*, que permitiu armazenar localmente os dados resultantes das requisições ao *Back-end*, evitando chamadas desnecessárias à API e garantindo maior fluidez na navegação entre páginas.

As respostas de algumas requisições críticas, como aquelas associadas à categorização de falhas, são também persistidas no *localStorage*, o que permite ao usuário visualizar os dados mesmo após navegar para outras seções e retornar posteriormente, sem perda de contexto.

3.4.6 Comunicação com a API

As requisições HTTP entre o *Front-end* e o *Back-end* foram implementadas utilizando a biblioteca *Axios*, que fornece uma interface simples para envio de dados, tratamento de erros e interceptação de respostas. A integração entre *Axios* e *React Query* foi fundamental para o desempenho e confiabilidade das interações entre cliente e servidor.

3.4.7 Visualização de Dados

A apresentação dos resultados das análises foi realizada com o uso da biblioteca *Chart.js*, responsável pela geração de gráficos dinâmicos e responsivos. Esses gráficos incluem representações como barras, linhas e setores, que facilitam a interpretação visual das métricas de confiabilidade e da categorização de falhas.

3.4.8 Validação de Formulários

A validação de formulários e envio de arquivos foi implementada por meio da biblioteca *react-hook-form*, em conjunto com o validador *zod*, possibilitando a definição de esquemas de validação declarativos. A combinação dessas ferramentas garantiu que os dados inseridos pelos usuários estivessem corretos antes de serem processados, inclusive com validações específicas para os arquivos exigidos pela plataforma, como *application.evtx* e *system.evtx*.

3.4.9 Autenticação

Para controle de acesso às rotas privadas da aplicação, foi utilizado o mecanismo de autenticação baseado em *JSON Web Tokens (JWT)*. A biblioteca *jwt-decode* permitiu a decodificação e verificação da validade dos tokens armazenados no *localStorage*, assegurando que apenas usuários autenticados tivessem acesso às seções restritas da plataforma.

3.5 Tecnologias Utilizadas no Back-end

O *Back-end* da plataforma X-RAT foi implementado em *Python 3.11*, devido à sua vasta coleção de bibliotecas para análise de dados, processamento de eventos e integração com banco de dados. A arquitetura foi organizada em torno da API *FastAPI*, que fornece

comunicação *RESTful* com o *Front-end*, validação automática de dados e documentação interativa (*Swagger/OpenAPI*). O banco de dados utilizado é o *PostgreSQL*, acessado via *SQLAlchemy ORM*, onde são armazenados os eventos extraídos e suas classificações.

3.5.1 Bibliotecas de Processamento de *Logs*

Para manipulação dos arquivos `.evtx`, que contêm os registros de falha do Windows, foram empregadas as bibliotecas:

1. *PyEvtxParser*: leitura e iteração sobre registros de eventos.
2. *lxml* e *xmltodict*: *parsing* eficiente de estruturas *XML*.
3. *re (expressões regulares)*: extração de padrões como *EventID*, *Channel* e *TimeCreated*.

3.5.2 Manipulação e Estruturação dos Dados

Os eventos extraídos são transformados em *DataFrames* utilizando a biblioteca *pandas*, permitindo operações como:

1. Normalização de campos (*SourceName*, *CauseCode*);
2. Definição e ajuste de *timezones* com *pytz*;
3. Agrupamento, categorização e contagem de falhas.

3.5.3 Análises Estatísticas

A análise de confiabilidade foi realizada com o auxílio da biblioteca *SciPy*, utilizando distribuições clássicas de confiabilidade:

1. *Weibull*, *Gamma*, *Lognormal*, *Normal* e *Exponencial*.

Foram aplicados testes de aderência (*Kolmogorov-Smirnov*,*Anderson-Darling*, *AIC*) para selecionar a melhor distribuição para cada conjunto de dados, em linha com trabalhos anteriores de caracterização de falhas em sistemas operacionais (SANTOS; MATIAS; TRIVEDI, 2020; SANTOS; JR.; TRIVEDI, 2021).

3.5.4 Containerização e *Deploy*

Para simplificar a execução local e o processo de implantação, a aplicação foi conteinerizada com *Docker*.

1. O *Dockerfile* define o ambiente *Python* com as dependências listadas em `requirements.txt`.

2. O *Docker Compose* orquestra os serviços de aplicação e banco de dados, permitindo que múltiplos usuários executem o sistema de forma consistente.

Essa estratégia reduz problemas de compatibilidade, acelera o processo de desenvolvimento em equipe e facilita o *deploy* em diferentes ambientes.

Capítulo 4

Resultados da Implementação

Este capítulo apresenta os resultados alcançados com o desenvolvimento da nova versão da plataforma X-RAT. Para contextualizar as melhorias implementadas, apresenta-se inicialmente uma comparação direta entre as funcionalidades da versão anterior e os avanços obtidos nesta atualização. Essa comparação permite visualizar de forma clara as limitações previamente existentes e as evoluções estruturais que possibilitaram uma análise mais precisa e automatizada das falhas registradas. A Tabela 2 sintetiza essas diferenças e destaca os principais aprimoramentos realizados.

Funcionalidade	**Primeira Versão do X-RAT**	**Nova Versão do X-RAT**
Coleta de logs de falhas	Não disponível – logs precisavam ser coletados manualmente	Disponível – Possui sistema de coleta automática e integrada dos logs de falhas
Mecanismo de filtro de eventos	Arquivo CSV externo, carregado pela aplicação	CRUD integrado na aplicação
Mecanismo de categorização de falhas	Arquivo CSV externo, carregado pela aplicação	CRUD integrado na aplicação
Autenticação (Login)	Não possuía sistema de *login* – acesso livre a qualquer usuário	Possui sistema de *login* integrado com autenticação de usuários para os mecanismos de filtro de eventos e categorização de falhas
Transformação de *timestamp*	Converte todos os *timestamps* automaticamente para fuso horário brasileiro (GMT-3)	Identifica o fuso horário real do computador no momento da falha e converte corretamente
Formato do relatório	Arquivo PDF estático gerado após o processamento	Relatório dinâmico, exibido diretamente na aplicação
Informações de causa de falhas	Não disponível – o sistema apenas registrava o evento de falha	Disponível – o sistema coleta e armazena as causas associadas às falhas

Tabela 2 – Comparação entre a primeira e a nova versão do X-RAT.

A seguir, apresenta-se o fluxo completo da plataforma X-RAT, descrevendo as etapas que abrangem desde a coleta dos arquivos de *log* do sistema operacional (`.evtx`) e o preenchimento do formulário de caracterização, até a geração das métricas de confiabilidade e a categorização automática das falhas. O objetivo é demonstrar como os dados são processados, transformados e utilizados para compor a análise de confiabilidade dos sistemas estudados.

4.1 Pré-processamento

A primeira etapa de acesso à plataforma X-RAT é o sistema de autenticação, desenvolvido exclusivamente para administradores. Essa funcionalidade garante que apenas usuários autorizados possam acessar as áreas de gerenciamento e realizar operações sensíveis, assegurando a integridade das informações armazenadas e o controle sobre os dados processados.

4.1.1 Registro de Usuário

O processo de registro permite a criação de novos usuários na plataforma, possibilitando o controle e a identificação individual de cada participante do sistema. Ao realizar o cadastro, o usuário é automaticamente inserido com a função (*role*) padrão de *usuário comum*. Essa categoria de acesso é restrita e destina-se exclusivamente às funcionalidades básicas, como a visualização e impressão de relatórios públicos ou demonstrativos gerados pela plataforma. Usuários com essa função não possuem permissão para autenticação no sistema administrativo, nem podem acessar áreas de gerenciamento de dados ou configuração. Para que um usuário possa efetuar login e ter acesso ao painel de administração, é necessário que um administrador altere manualmente sua função para o nível de *administrador*. Essa medida garante maior segurança e controle sobre os acessos ao sistema, evitando o uso indevido das funcionalidades administrativas. A Figura 5 apresenta a tela de registro de novos usuários, onde é possível visualizar os campos obrigatórios para o cadastro.

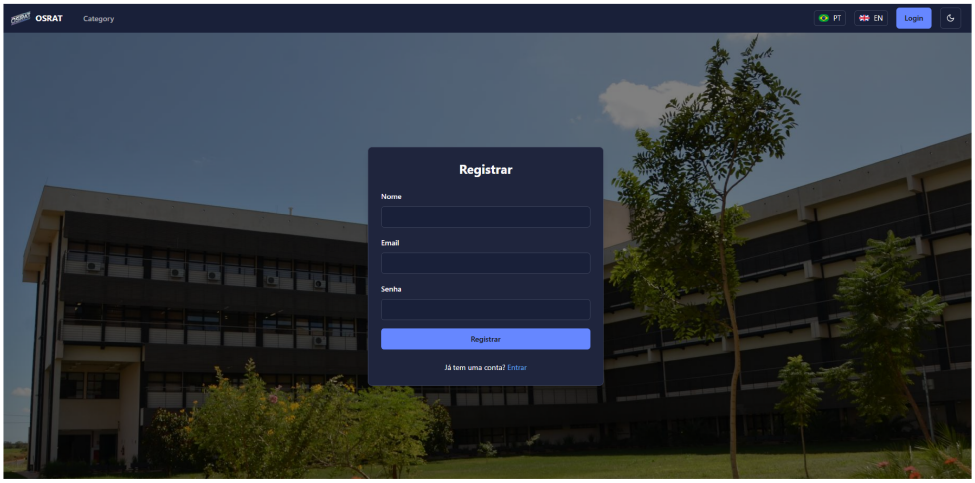


Figura 5 – Tela de registro de novo usuário.

4.2 Gerenciamento de Usuários

O módulo de gerenciamento de usuários tem como finalidade permitir que administradores controlem as permissões de acesso à plataforma. Sempre que um novo usuário se registra, ele é automaticamente classificado como *usuário comum*. Apenas administradores podem alterar a função de um usuário, promovendo-o para o perfil de administrador quando necessário. Essa funcionalidade garante a segurança do sistema e possibilita que múltiplos administradores trabalhem de forma colaborativa no gerenciamento das informações, sem comprometer o controle de acesso.

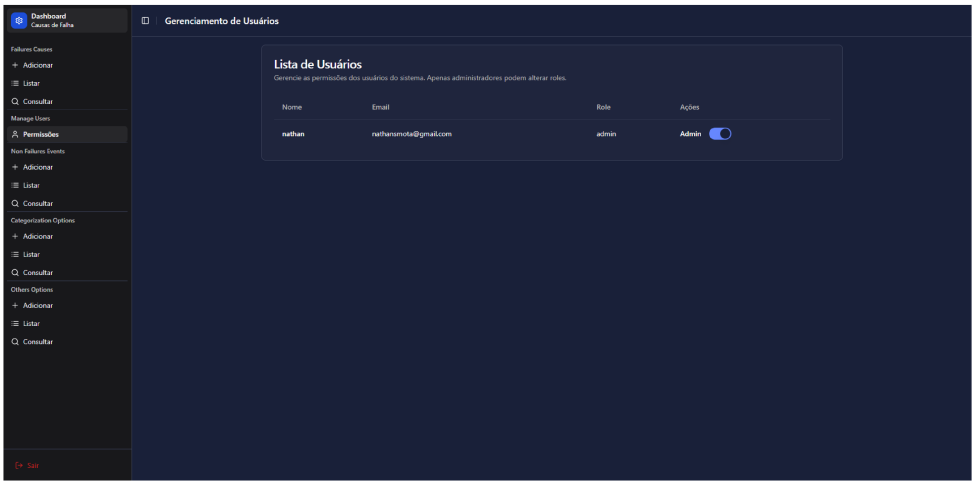


Figura 6 – Tela de gerenciamento de usuários e alteração de permissões.

A Figura 6 apresenta a tela de gerenciamento de usuários, onde é possível alterar permissões e acompanhar os perfis cadastrados.

4.2.1 *Login* de Usuário

Após o registro, o usuário autorizado pode efetuar o *login* utilizando suas credenciais. A autenticação é realizada de forma segura, validando o nome de usuário, senha e função de acesso. Essa verificação garante que apenas perfis com a *role* de administrador possam acessar o painel de gerenciamento da plataforma. A Figura 7 ilustra a tela de autenticação do sistema, onde ocorre a validação das credenciais de acesso.

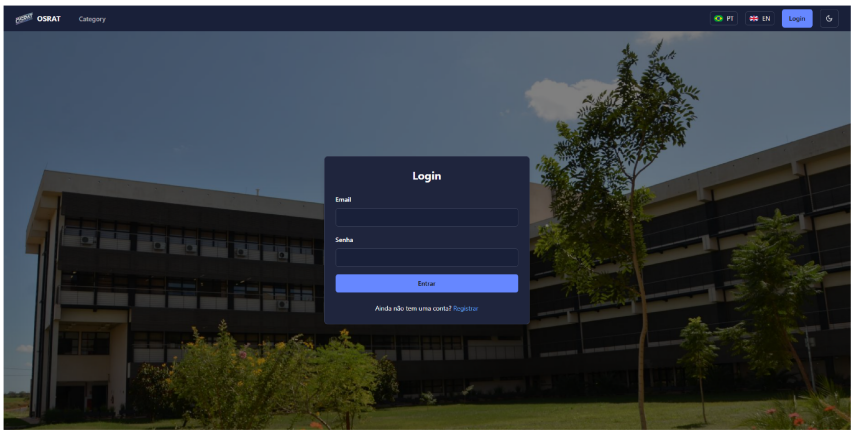


Figura 7 – Tela de autenticação do sistema.

4.2.2 Painel de Administração

Após a autenticação, o usuário administrador tem acesso ao painel de controle do sistema, onde é possível visualizar e gerenciar as principais entidades utilizadas na plataforma.

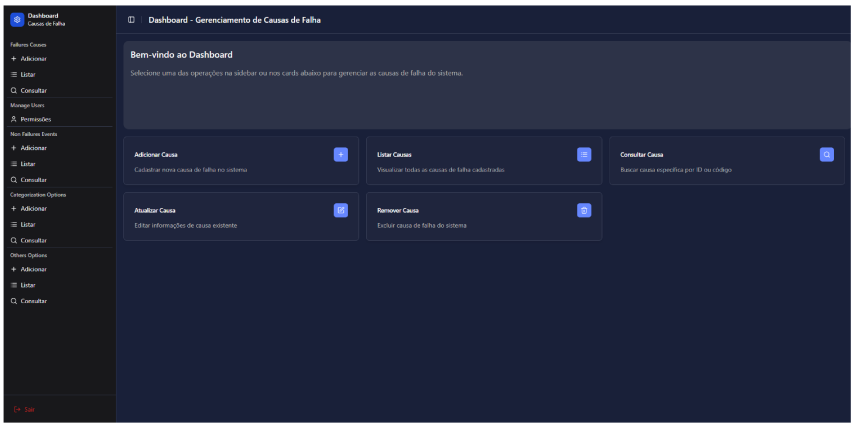


Figura 8 – Painel de gerenciamento administrativo da plataforma.

O painel apresenta opções organizadas de acordo com o tipo de dado, incluindo falhas, eventos, aplicativos e usuários. A Figura 8 mostra o painel de gerenciamento administrativo da plataforma, destacando as principais seções disponíveis para o administrador.

4.3 Gerenciamento de Causas de Falhas

O módulo de causas de falhas é um dos principais componentes administrativos, permitindo o controle completo das informações relacionadas às falhas identificadas. Esse módulo foi implementado com operações de *CRUD* (*Create, Read, Update, Delete*), que possibilitam a inclusão, edição, exclusão e consulta de registros. A Figura 9 apresenta o formulário de cadastro de uma nova causa de falha, no qual o administrador pode registrar a fonte (*SourceName*) — que indica o componente, *driver*, serviço, aplicação ou subsistema do Windows responsável pela geração do evento —, o código hexadecimal da causa e a respectiva descrição. A descrição, obtida a partir da documentação oficial da Microsoft, permite ao sistema identificar e exibir automaticamente a descrição da causa sempre que o mesmo código for detectado. A Figura 10 exibe a interface destinada à listagem, edição e exclusão de causas de falhas. Por fim, a Figura 11 mostra a tela de consulta detalhada de uma falha específica, possibilitando a visualização completa dos dados armazenados para cada ocorrência.

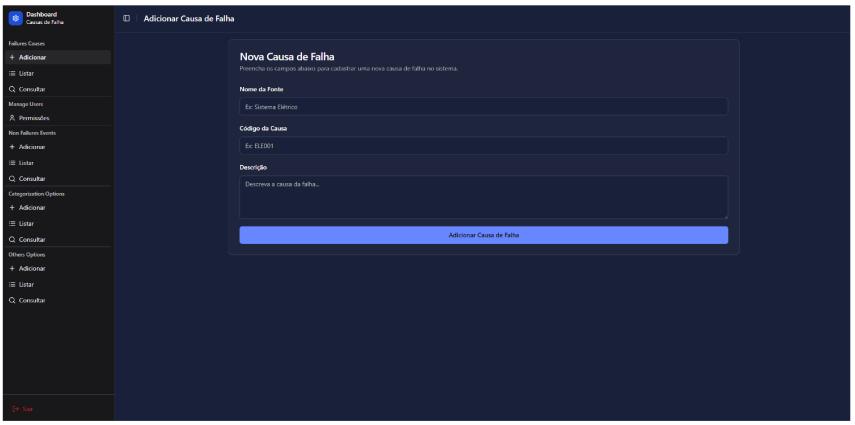


Figura 9 – Formulário de cadastro de nova causa de falha.

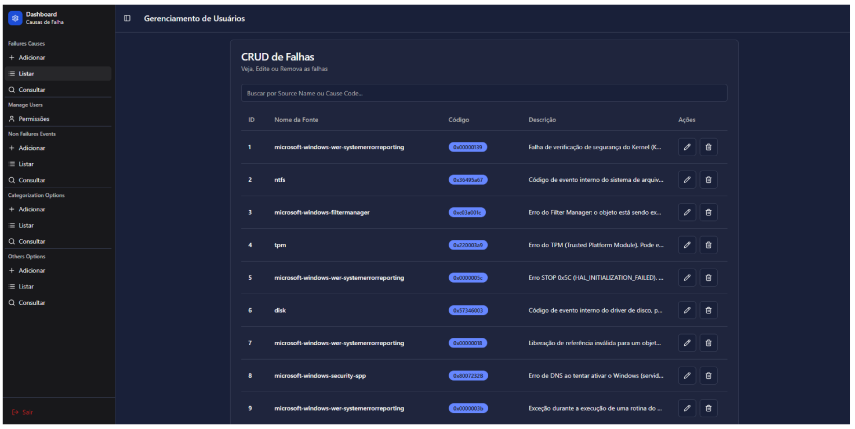


Figura 10 – Interface para listagem, edição e exclusão de causas de falhas.

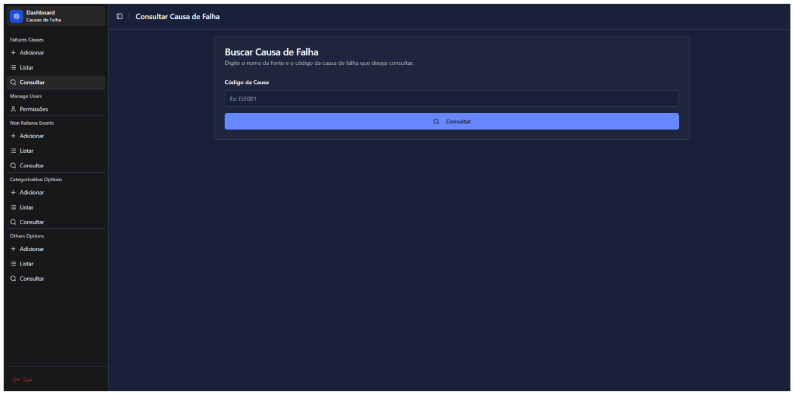


Figura 11 – Tela de consulta detalhada de uma falha específica.

4.4 Critérios de categorização das falhas

Este módulo, disponível para administradores, é responsável pela definição dos critérios de categorização dos eventos. Os *SourceNames* considerados nesta categorização foram: *windowsupdateclient*, *wer-systemerrorreporting*, *application hang* e *application error*. Com base nos critérios adicionados é determinado se o evento pertence às categorias OS_{KNL}, OS_{SVC}, OS_{APP} ou USR_{APP}. As Figuras 12, 13 e 14 ilustram, respectivamente, as telas de cadastro, listagem e consulta de aplicações registradas no sistema.

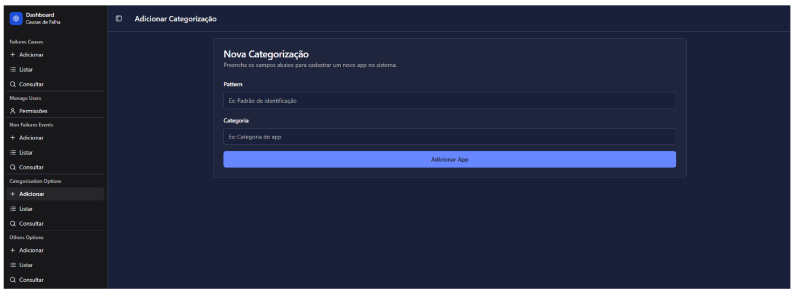


Figura 12 – Cadastro de um novo critério de categorização.

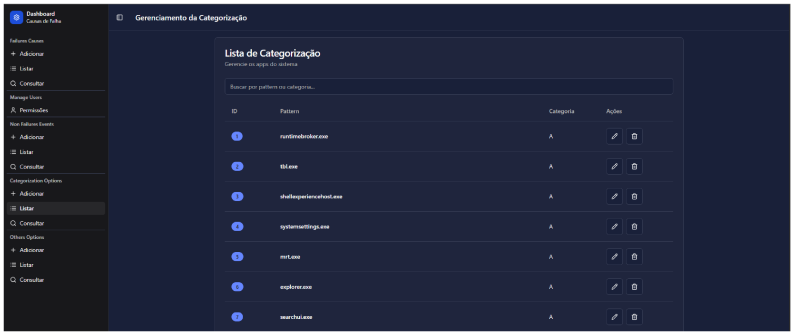


Figura 13 – Listagem e gerenciamento dos critérios registrados.

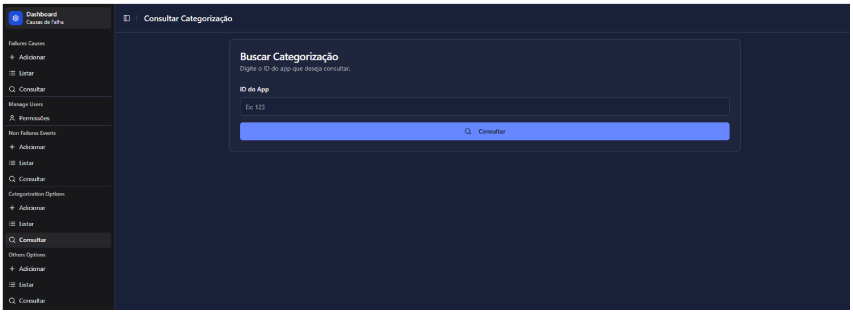


Figura 14 – Consulta de um critério específico.

Um módulo adicional foi incluído para classificar falhas que não estão nas principais fontes de falhas (*windowsupdateclient*, *wer-systemerrorreporting*, *application hang* e *application error*). As Figuras 15, 16 e 17 apresentam, respectivamente, as interfaces de cadastro, listagem e consulta dessas falhas.

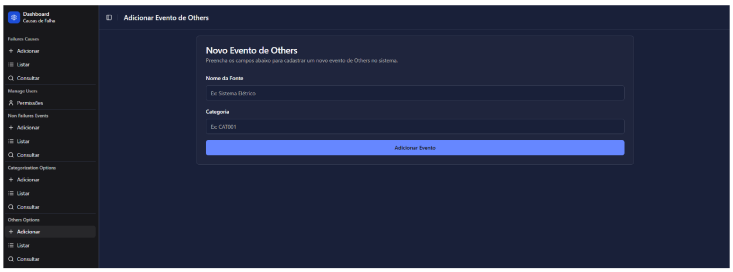


Figura 15 – Cadastro de um novo critério de categorização para falhas menos frequentes.

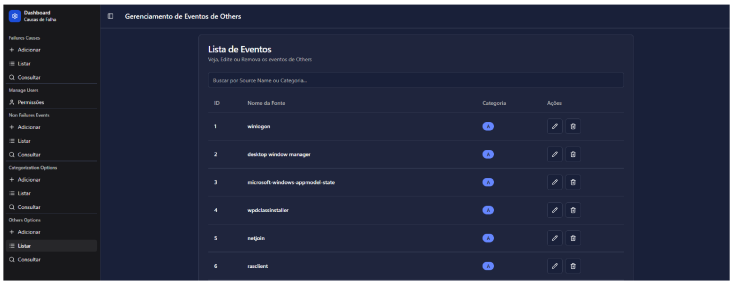


Figura 16 – Listagem e gerenciamento dos critérios registrados para falhas menos frequentes.

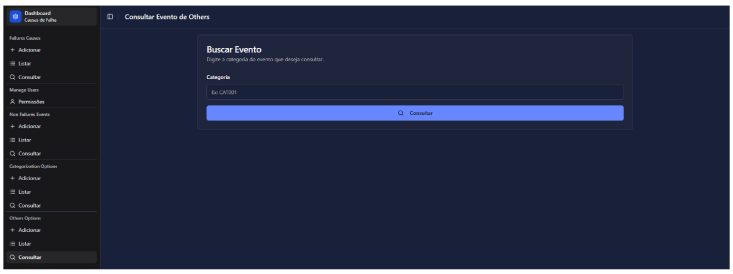


Figura 17 – Consulta de um critério específico para falhas menos frequentes.

4.5 Eventos não caracterizados como falhas

Neste módulo são registrados os eventos que não são caracterizados como falhas. Durante o processo de extração dos eventos a partir dos arquivos *.evtx*, caso um evento pertença a essa lista, ele é automaticamente excluído da importação. Esse módulo segue a mesma estrutura de *CRUD*, permitindo o controle completo dos registros.

A Figura 18 apresenta a tela de cadastro de um novo evento sem falha, enquanto as Figuras 19 e 20 mostram, respectivamente, a listagem geral e a consulta individual de um evento.

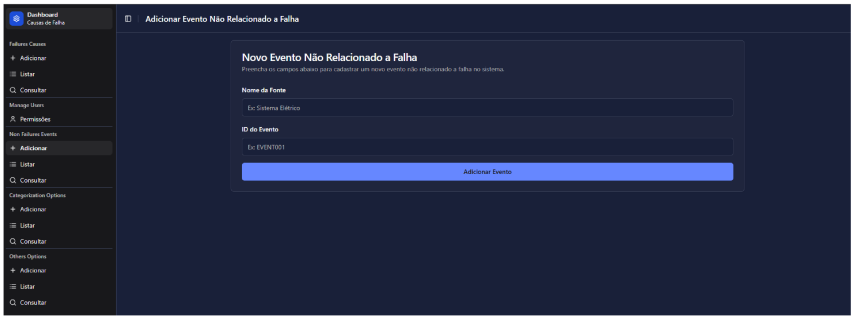


Figura 18 – Cadastro de novo evento não caracterizado como falha.

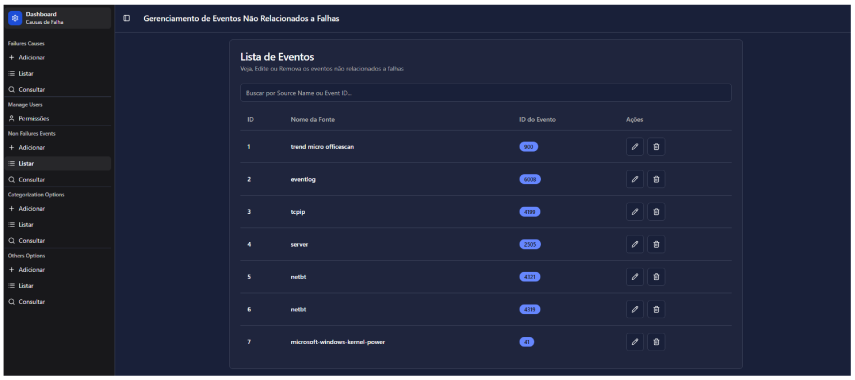


Figura 19 – Listagem e gerenciamento de eventos não relacionados a falhas.

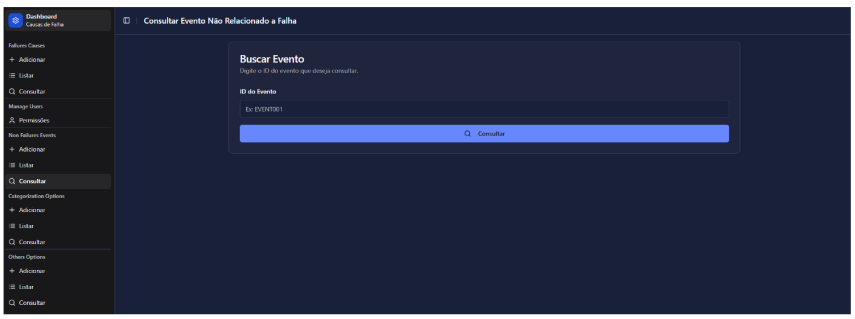


Figura 20 – Consulta individual de evento não caracterizado como falha.

4.6 Processamento dos Logs de Falha

Esta seção apresenta o fluxo completo de interação do usuário com a plataforma X-RAT, desde o primeiro acesso até a visualização dos resultados das análises de confiabilidade. O objetivo é demonstrar, de forma prática, o processo de participação dos usuários na pesquisa, bem como as etapas de envio dos arquivos de log e preenchimento do formulário de caracterização.

Ao acessar a plataforma, o usuário é recebido pela tela inicial apresentada na Figura 21. Essa interface fornece informações introdutórias sobre o propósito da pesquisa, esclarecendo que o estudo tem como foco a análise de falhas em sistemas operacionais Windows 10 e Windows 11, com ênfase na confiabilidade de software. Também é informado que a contribuição do participante ocorre exclusivamente por meio do envio dos arquivos de *log* e do preenchimento do formulário de caracterização, garantindo total privacidade dos dados.

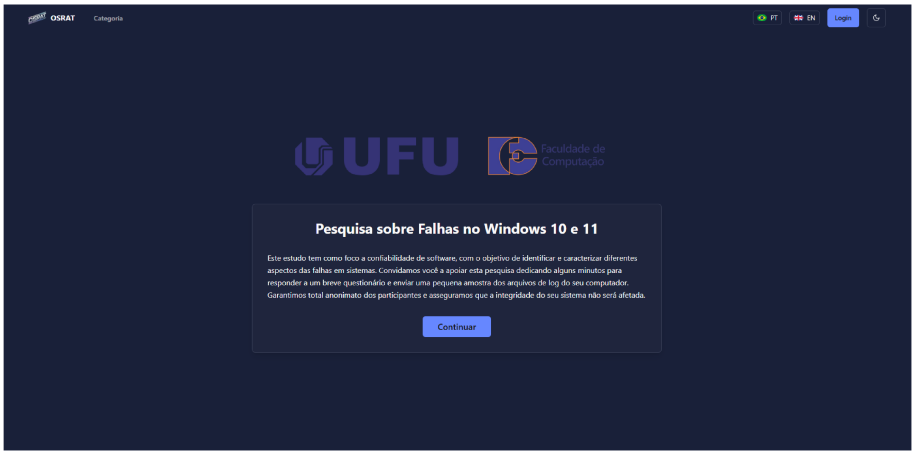


Figura 21 – Tela inicial de apresentação da pesquisa.

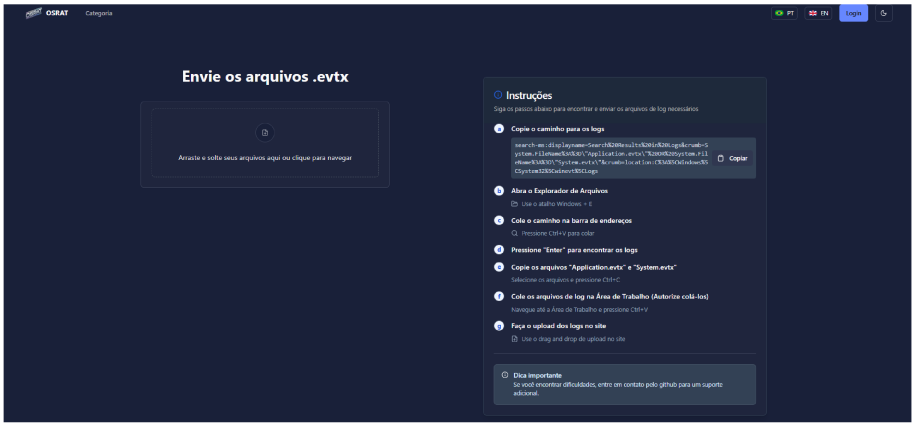


Figura 22 – Interface de *upload* de arquivos de *log* do sistema.

Ao clicar em *Continuar*, o usuário é direcionado à primeira etapa do processo, que consiste na interface de envio dos arquivos de *log* (*upload*), ilustrada na Figura 22. Nessa etapa, é iniciado o processo de coleta dos dados que servirão de base para as análises de confiabilidade. O usuário é orientado a localizar os arquivos `Application.evtx` e `System.evtx`, responsáveis por registrar eventos e falhas do sistema operacional. Para facilitar essa tarefa, a plataforma disponibiliza um comando que pode ser copiado e inserido diretamente no explorador de arquivos do Windows, conforme apresentado a seguir:

```
search-ms:displayname=Search%20Results%20in%20Logs&crumb=System.FileName%3A%3D"
    Application.evtx"%20OR%20System.FileName%3A%3D"System.evtx"&crumb=location:C
    %3A%5CWindows%5CSystem32%5Cwinevt%5CLogs
```

Ao clicar no botão “Copiar”, o comando é automaticamente copiado para a área de transferência, sendo exibida uma mensagem de confirmação, conforme mostrado na Figura 23.

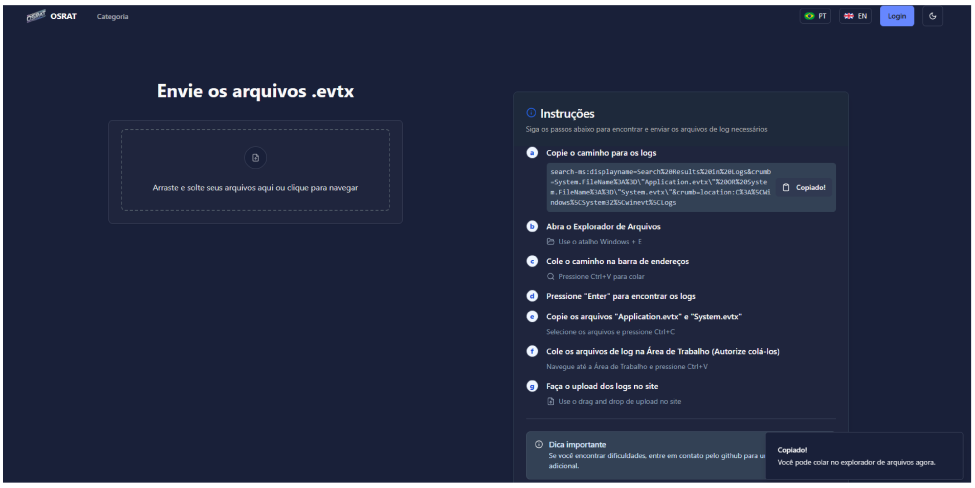


Figura 23 – Confirmação da cópia do comando de busca dos arquivos de *log*.

Ao colar o comando no menu de busca de arquivos do Windows, o usuário tem acesso direto à pasta onde os *logs* estão armazenados, como ilustrado na Figura 24.

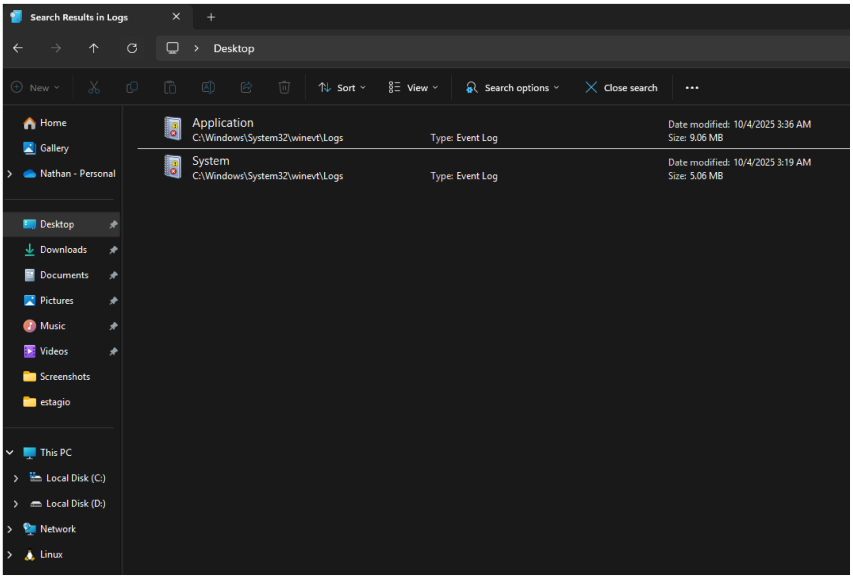


Figura 24 – Localização dos arquivos de *log* do Windows.

Por motivos de permissão do sistema, não é possível enviar os arquivos diretamente da pasta original. Assim, é necessário mover os arquivos localizados para a área de trabalho (*Desktop*), conforme mostra a Figura 25. Essa etapa evita restrições de acesso e garante o sucesso do upload.

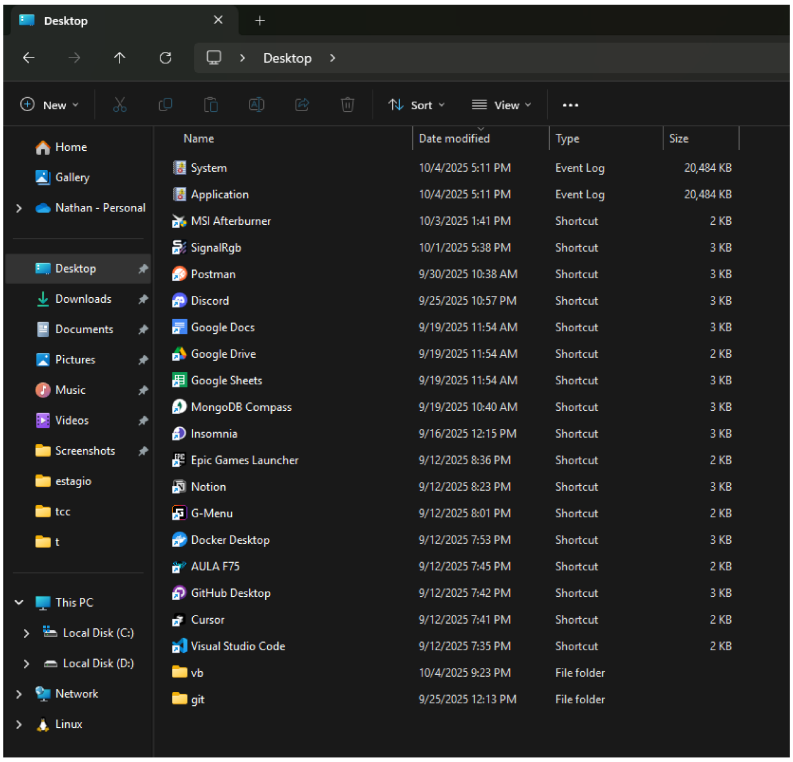


Figura 25 – Movimentação dos arquivos de *log* para o *Desktop*.

Após mover os arquivos, o usuário pode arrastá-los (*drag and drop*) diretamente para a área de upload da plataforma, conforme exemplificado na Figura 26.

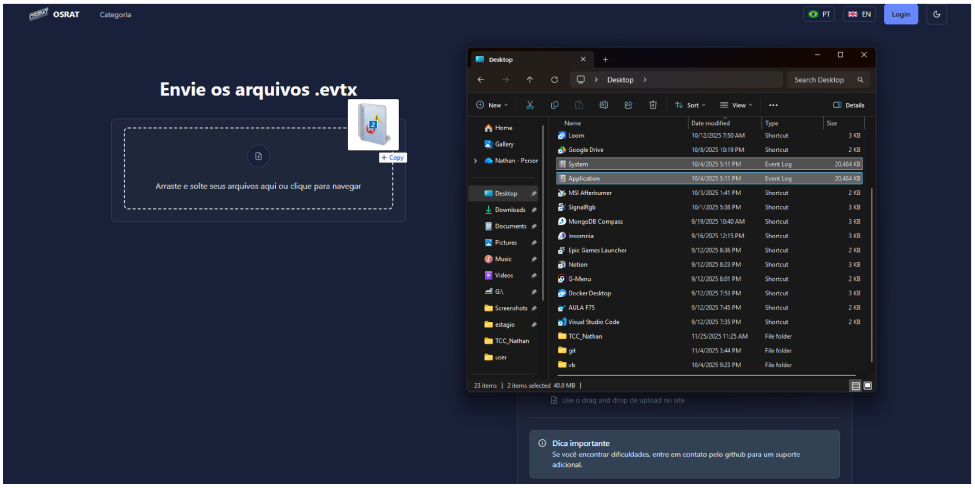


Figura 26 – Envio dos arquivos `.evtx` para análise.

Na segunda etapa do processo, o usuário deve preencher o formulário de caracterização, ilustrado na Figura 27. Esse formulário tem como objetivo coletar informações contextuais sobre o ambiente em que o computador opera, permitindo análises mais precisas e comparativas entre diferentes perfis de uso.

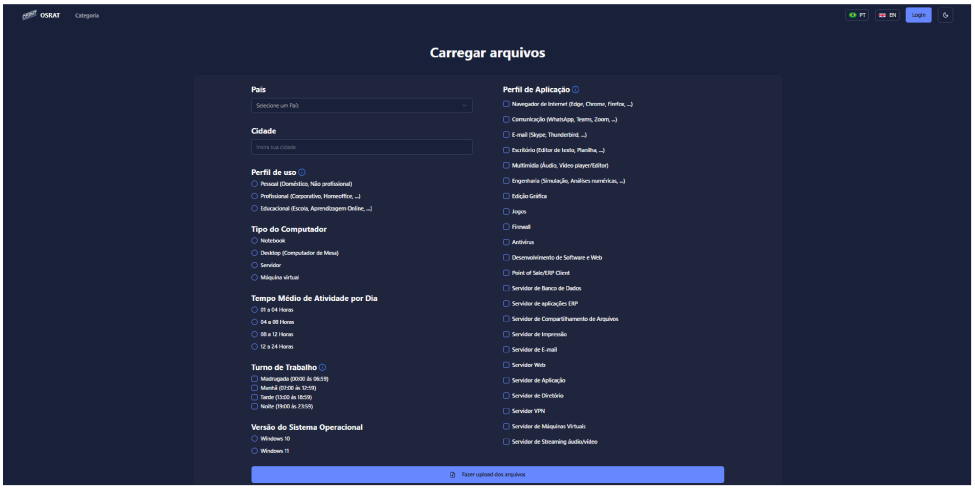


Figura 27 – Formulário de caracterização do ambiente de uso.

As informações solicitadas incluem:

1. Tipo de ambiente (laboratorial, corporativo ou pessoal);
2. Tipo de equipamento utilizado;
3. Tempo médio de uso diário;
4. Horários mais frequentes de uso;
5. Versão do sistema operacional;

6. Perfil das aplicações mais utilizadas.

Esses dados complementares são fundamentais para contextualizar os resultados e identificar padrões de falhas específicos a determinados cenários de uso. Após o envio do formulário e dos arquivos de *log*, o *Back-end* da plataforma — desenvolvido em Python utilizando o *framework* FastAPI — executa automaticamente as etapas de extração, categorização e análise dos eventos. O sistema gera diversas métricas de confiabilidade, entre as quais se destacam:

1. Número total de falhas detectadas;
2. Distribuição de falhas por dia da semana;
3. Distribuição de falhas por período do dia (manhã, tarde, noite e madrugada);
4. Classificação das falhas por tipo (*kernel*, serviços, aplicativos do sistema e aplicativos do usuário);
5. Identificação de falhas críticas com base em códigos específicos de erro.

O resultado dessa análise é apresentado ao usuário na forma de um relatório completo, acompanhado de gráficos e tabelas que sintetizam o comportamento das falhas. A Figura 28 ilustra as distribuições de falhas por período do dia e por dia da semana.

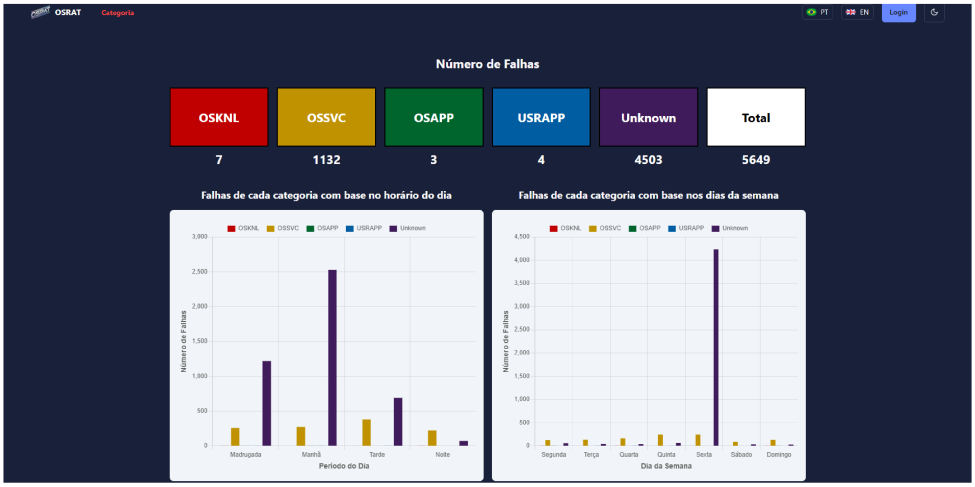
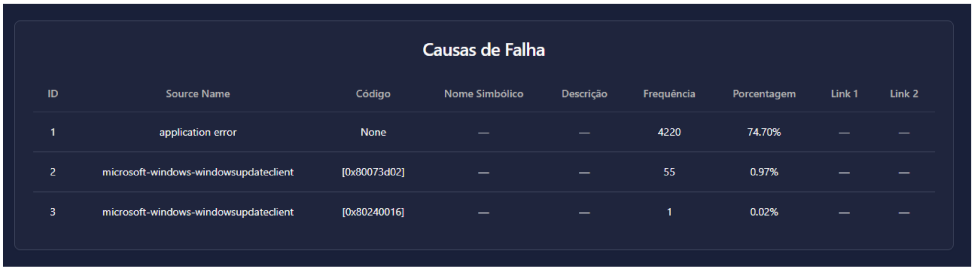


Figura 28 – Distribuição temporal das falhas por período e por dia da semana.

A Figura 29 apresenta a tabela de causas de falhas, contendo colunas como *Source Name*, código de evento, frequência e porcentagem de ocorrência em relação ao total de falhas observadas.



Causas de Falha

ID	Source Name	Código	Nome Simbólico	Descrição	Frequência	Porcentagem	Link 1	Link 2
1	application error	None	—	—	4220	74.70%	—	—
2	microsoft-windows-windowsupdateclient	[0x80073d02]	—	—	55	0.97%	—	—
3	microsoft-windows-windowsupdateclient	[0x80240016]	—	—	1	0.02%	—	—

Figura 29 – Tabela de causas de falhas com frequência e percentual de ocorrência.

Em seguida, a Figura 30 mostra as estatísticas gerais de ocorrência das falhas, abrangendo medidas de tendência central e dispersão, como média, mediana, valores mínimo e máximo, entre outras, permitindo uma análise descritiva da confiabilidade do sistema.

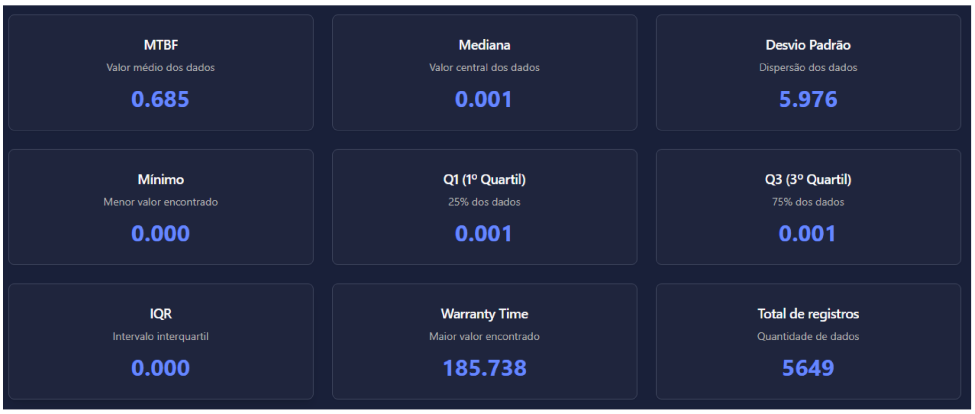


Figura 30 – Estatísticas dos tempos entre as falhas (em horas).

A Figura 31 representa o histograma de tempo entre falhas (*Time Between Failures — TBF*), expresso em horas. Esse gráfico possibilita observar a distribuição de frequência das ocorrências e fornece uma base visual para o cálculo de métricas como o MTBF (*Mean Time Between Failures*).

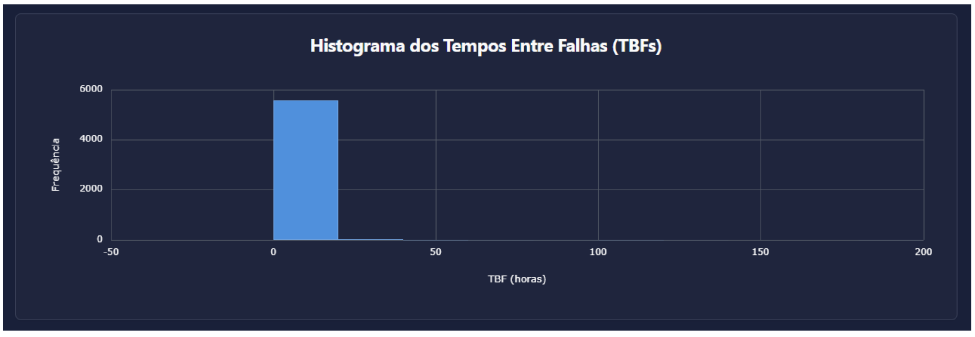


Figura 31 – Histograma do tempo entre falhas (TBF).

Figura 32 – Resultados do teste de aderência (*Goodness-of-Fit*) e gráficos de confiabilidade.

A Figura 32 apresenta os resultados do *Goodness-of-Fit Test*, responsável por avaliar o ajuste dos dados às principais distribuições estatísticas utilizadas em estudos de confiabilidade, como Lognormal, Weibull, Gamma, Exponencial e Normal. A tabela exibida é dinâmica e permite a seleção da distribuição desejada, atualizando automaticamente os gráficos correspondentes, que ilustram o comportamento da confiabilidade do sistema.

Complementando essa análise, as Figuras 33 e 34 apresentam, respectivamente, a taxa de falhas $h(t)$ (ou *failure rate*) e a função densidade de probabilidade $f(t)$ (PDF) associada aos tempos entre falhas (TBF). Esses gráficos possibilitam observar de forma mais detalhada o comportamento temporal das falhas e a distribuição de sua ocorrência, oferecendo uma visão mais completa do modelo de confiabilidade ajustado.

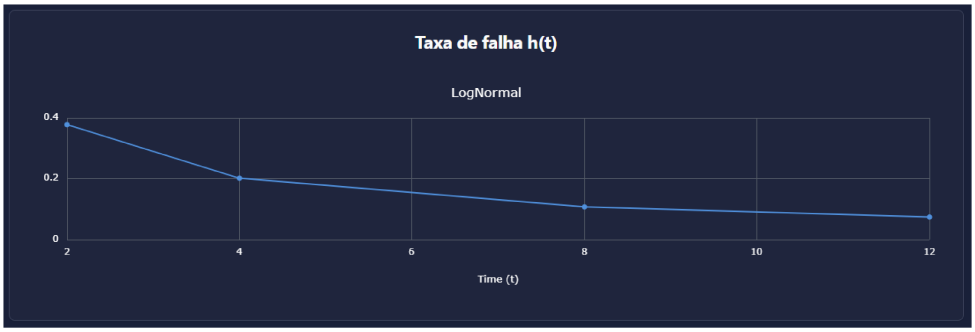


Figura 33 – Taxa de falhas (*failure rate*) $h(t)$ calculada a partir da distribuição ajustada.

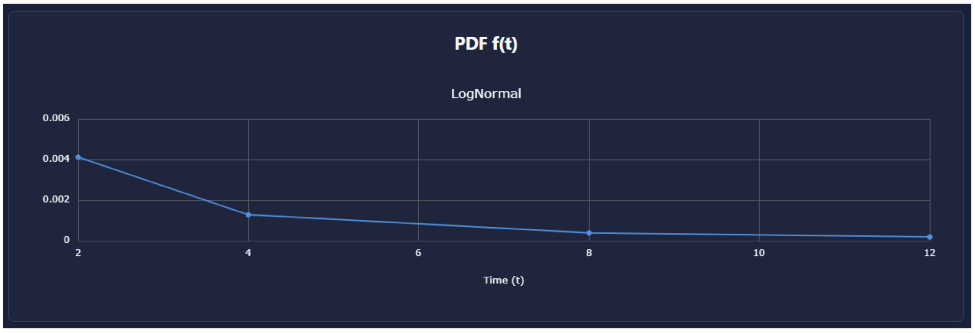


Figura 34 – Função densidade de probabilidade (PDF) $f(t)$ dos tempos entre falhas (TBF).

4.7 Interface e Internacionalização

A plataforma X-RAT foi desenvolvida com foco na acessibilidade e na experiência do usuário, oferecendo suporte à personalização visual e linguística. O sistema permite alternar entre os modos claro e escuro, atendendo a diferentes preferências de uso e condições de luminosidade do ambiente. Além disso, a interface possui suporte à internacionalização (*i18n*), permitindo a alternância dinâmica entre os idiomas português e inglês. Essa funcionalidade torna a plataforma acessível a um público mais amplo e facilita o uso em contextos acadêmicos e corporativos internacionais. As Figuras 35 e 36 ilustra esses recursos, apresentando a tela inicial da plataforma no modo claro, exibida em dois idiomas distintos: português e inglês. Esse recurso demonstra a capacidade de adaptação da interface sem alterar a estrutura visual da aplicação.

(a) Interface em modo claro - PT-BR

Figura 35 – Interface inicial da plataforma X-RAT no modo claro, com suporte à internacionalização em PT-BR.

(b) Interface em modo claro - EN

Figura 36 – Interface inicial da plataforma X-RAT no modo claro, com suporte à internacionalização em EN.

4.8 Avaliação da Qualidade da Interface

A nova interface da plataforma X-RAT foi projetada com foco em usabilidade, acessibilidade e aderência às boas práticas modernas de desenvolvimento *web*. Para validar essas melhorias de forma objetiva, realizou-se uma análise utilizando a ferramenta *Lighthouse*, amplamente empregada para avaliar critérios de qualidade relacionados à experiência do usuário, acessibilidade e conformidade técnica.

Os resultados obtidos demonstram que a interface atinge elevados padrões de qualidade, apresentando 94% em *Accessibility* (Acessibilidade) e 93% em *Best Practices* (Boas Práticas). Esses valores confirmam que a refatoração proposta não apenas moderniza a apresentação da plataforma, mas também garante maior conformidade com diretrizes internacionais de acessibilidade e boas práticas de engenharia de software.

A Figura 37 ilustra a avaliação realizada, evidenciando os indicadores alcançados pela aplicação após a refatoração.

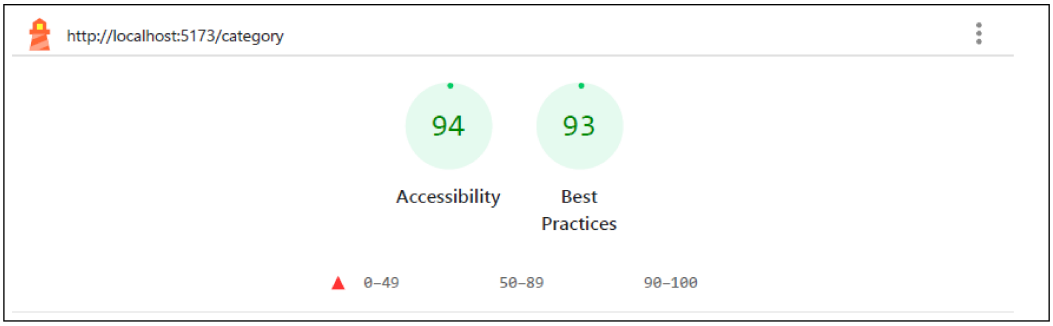


Figura 37 – Avaliação da interface da plataforma X-RAT utilizando o *Lighthouse*, apresentando 94% em *Accessibility* (Acessibilidade) e 93% em *Best Practices* (Boas Práticas).

Esses resultados reforçam que a nova versão da plataforma proporciona uma experiência superior ao usuário, com melhor estruturação dos elementos visuais, navegação mais intuitiva e aderência a práticas recomendadas para aplicações modernas. Além disso, demonstram que a interface refatorada está alinhada com padrões que favorecem inclusão, robustez e manutenibilidade.

Capítulo 5

Considerações finais e Trabalhos futuros

Este capítulo apresenta as considerações finais sobre o desenvolvimento e os resultados obtidos com a plataforma X-RAT, relacionando-os aos objetivos propostos e às contribuições alcançadas ao longo da pesquisa. A seguir, são discutidos como os objetivos geral e específicos foram atingidos, as principais contribuições deste trabalho e as perspectivas de aprimoramento para estudos futuros.

A proposta central deste trabalho consistiu em aprimorar a plataforma X-RAT, originalmente desenvolvida em trabalhos anteriores (PEIXOTO, 2023; LOPES, 2023), ampliando sua capacidade de análise de confiabilidade de *software* por meio da implementação da funcionalidade de identificação das causas de falhas em sistemas operacionais. Esse aprimoramento permitiu transformar a ferramenta em uma solução completa, capaz de coletar, processar e analisar automaticamente eventos de falhas, gerando relatórios detalhados de confiabilidade em computadores reais.

Durante o desenvolvimento, o objetivo geral — `investigar as causas de falhas em computadores que operam com os sistemas Windows 10 e 11, utilizando uma abordagem automatizada de análise de confiabilidade de software` — foi plenamente atingido. A refatoração da plataforma e o desenvolvimento *Full Stack* (*Front-end* e *Back-end*) viabilizaram a integração de todas as etapas do processo, desde a coleta dos arquivos de *log*, até a categorização das falhas e geração automática dos relatórios. Com isso, foi possível realizar análises consistentes sobre os eventos de falha, alinhadas às categorias propostas em estudos clássicos sobre confiabilidade de sistemas operacionais (JR. et al., 2013; JR. et al., 2014; SANTOS; JR.; TRIVEDI, 2021).

Kernel ($\mathrm{OS_{KNL}}$), Serviços do Sistema ($\mathrm{OS_{SVC}}$) e Aplicações do Sistema ($\mathrm{OS_{APP}}$) e aplicações do usuário ($\mathrm{USR_{APP}}$)

Os objetivos específicos também foram alcançados de forma satisfatória:

- **Implementar uma metodologia de identificação e categorização das causas de falhas de *software***: a categorização foi estruturada de acordo com as três classes principais — *Kernel* ($\mathrm{OS_{KNL}}$), Serviços do Sistema ($\mathrm{OS_{SVC}}$) e Aplicações do

Sistema (OS_{APP}) — conforme a taxonomia proposta por Matias (JR. et al., 2013) e consolidada em trabalhos posteriores (JR. et al., 2014; SANTOS; JR.; TRIVEDI, 2021). Essa estrutura possibilitou a associação de eventos a suas respectivas causas de falhas, tornando o diagnóstico mais preciso e fundamentado.

- ❑ **Desenvolver o *Front-end* e *Back-end* integrados da plataforma**: o *Back-end*, implementado em *FastAPI*, e o *Front-end*, desenvolvido em *React* com suporte à internacionalização, proporcionaram uma interface intuitiva e acessível, permitindo que usuários não especializados possam enviar seus arquivos de *log* e obter relatórios detalhados automaticamente.

- ❑ **Gerar relatórios automáticos e métricas de confiabilidade**: a plataforma calcula e apresenta métricas como MTBF (*Mean Time Between Failures*) e taxa de falhas, além de estatísticas sobre a distribuição dos eventos, permitindo uma visão quantitativa da confiabilidade dos sistemas analisados.

Dessa forma, os resultados obtidos demonstram que a pesquisa atingiu seus objetivos, oferecendo uma ferramenta eficaz para análise de confiabilidade de sistemas operacionais, alinhada com as abordagens empíricas e exploratórias descritas na literatura (JR. et al., 2013; JR. et al., 2014; SANTOS; JR.; TRIVEDI, 2021; AVIZIENIS et al., 2004).

As principais contribuições deste trabalho concentram-se em três eixos: técnico, metodológico e científico.

- ❑ **Contribuição técnica**: desenvolvimento de uma plataforma *web Full Stack* moderna e modular, que automatiza o processo de análise de confiabilidade de sistemas operacionais, integrando coleta, processamento e visualização de dados em uma única solução. Essa integração facilita o uso em ambientes laboratoriais, corporativos ou pessoais, ampliando o alcance e aplicabilidade do X-RAT.

- ❑ **Contribuição metodológica**: implementação de um *pipeline* completo de análise baseado em dados reais, utilizando *logs* de eventos (`.evtx`) do Windows para identificar e classificar falhas segundo uma taxonomia consolidada na literatura de confiabilidade (AVIZIENIS et al., 2004; JR. et al., 2013; JR. et al., 2014). Essa abordagem automatizada reduz a subjetividade e o esforço manual na análise, contribuindo para a reprodutibilidade dos experimentos.

- ❑ **Contribuição científica**: consolidação da plataforma X-RAT como uma ferramenta de apoio à pesquisa em confiabilidade de *software*, permitindo a ampliação dos estudos sobre causas de falhas em sistemas operacionais de propósito geral, com base em grandes volumes de dados reais. Os resultados obtidos validam a hipótese de que a maioria das falhas observadas decorre de componentes de serviços do sistema (OS_{SVC}), corroborando estudos anteriores (JR. et al., 2014; SANTOS; JR.; TRIVEDI, 2021).

Com isso, este trabalho contribui para a continuidade das pesquisas do grupo X-RAT na Universidade Federal de Uberlândia, fortalecendo o arcabouço experimental existente e fornecendo uma base para análises mais amplas de confiabilidade e disponibilidade em sistemas computacionais. Como trabalhos futuros, pretende-se aprimorar a plataforma X-RAT em dois eixos principais, conforme orientação recebida:

- **Expansão do conjunto de eventos de falhas classificados**: aumentar a base de dados de eventos e causas de falhas para incluir novas versões de sistemas operacionais e tipos de eventos ainda não classificados, visando melhorar a precisão e a abrangência da categorização.
- **Análise de grupos de computadores**: incluir uma funcionalidade que permita comparar métricas de confiabilidade entre diferentes grupos de sistemas (por exemplo, ambientes corporativos, educacionais e domésticos), possibilitando a avaliação de padrões coletivos de falhas e tendências de confiabilidade em larga escala.

Referências

ABBAS, A.; KHAN, M.; ALI, S. The relationship between computers and society: Impacts, challenges, and opportunities. **Journal of Computer Science**, 2023. Disponível em: <https://www.researchgate.net/publication/376642881>.

AL-KILIDAR, H.; COX, K.; KITCHENHAM, B. The use and usefulness of the iso/iec 9126 quality standard. In: IEEE. **2005 International Symposium on Empirical Software Engineering, 2005.** [S.l.], 2005. p. 7–pp.

ANSI/IEEE. Standard, **Standard Glossary of Software Engineering Terminology**. New York, NY, USA: The Institute of Electrical and Electronics Engineers, Inc., 1991. Revisão e redesignação de IEEE Std 729-1983. Aprovado em 1990, muitas vezes citado como 1991.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. **IEEE Transactions on Dependable and Secure Computing**, v. 1, p. 11–33, Oct 2004.

BENNETT, J. V.; BRACHMAN, P. S. **Hospital Infections**. 6. ed. [S.l.]: Lippincott Williams & Wilkins, 2010. Citado em Woeltje (Use of Computerized Systems in Healthcare Epidemiology).

BRASIL. **Lei nº 13.709, de 14 de agosto de 2018. Lei Geral de Proteção de Dados Pessoais (LGPD)**. 2018. <http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/L13709.htm>. Acesso em: set. 2025.

ESPOSITO, F.; MASTROIANNI, C. Information technology and personal computers: The relational life cycle. In: **Proceedings of the International Conference on Information Systems**. [S.l.: s.n.], 2002.

GURUNG, B. **An Empirical Evaluation of Vite against Create React App**. 2022. <https://vitejs.dev/>. Acesso em: jul. 2025.

INSTITUTE, U. **Annual Global Data Center Survey 2021**. 2021.

JR., R. M. et al. Operating system reliability from the quality of experience viewpoint: An exploratory study. **Proceedings of the Brazilian Symposium on Computing Systems Engineering (SBESC)**, 2013.

JR., R. M. et al. An empirical exploratory study on operating system reliability. In: **Proceedings of the ACM Symposium on Applied Computing (SAC)**. [S.l.: s.n.], 2014. p. 1–8.

LOPES, B. C. **Geração Automática de Relatórios de Confiabilidade: Previsão de falhas de software com base nos padrões de eventos múltiplos na Plataforma X-RAT**. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Uberlândia, Brasil, dezembro 2023.

LYU, M. R. et al. **Handbook of software reliability engineering**. [S.l.]: IEEE computer society press Los Alamitos, 1996. v. 222.

MARTIN, J. Reliability of systems in critical infrastructure. **Software Engineering Journal**, v. 45, p. 123–135, 2023.

MARTINO, R. **Software Reliability Engineering: Principles and Practice**. [S.l.]: Elsevier, 2014.

METZ, J. **Windows XML Event Log (EVTX) format specification**. 2023. <https://github.com/libyal/libevtx/blob/main/documentation/Windows%20XML%20Event%20Log%20(EVTX).asciidoc>. Acesso em: set. 2025.

Microsoft. **Bug Check Code Reference**. 2021. Acesso em: 02 out. 2025. Disponível em: <https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/bug-check-code-reference2?redirectedfrom=MSDN>.

MICROSOFT. **Troubleshooting NetBT Error 4319**. 2021. Acesso em: 02 out. 2025. Disponível em: <https://learn.microsoft.com/en-us/answers/questions/357668/troubleshooting-netbt-error-4319>.

Microsoft. **Informações de Ciclo de Vida do Windows 10 e Windows 11**. 2023. <https://learn.microsoft.com/pt-br/lifecycle/products/>. Acesso em: set. 2025.

OLIVEIRA, B. S. **Análise da Confiabilidade do Sistema Operacional Windows 10 com a Ferramenta OSRat**. Dissertação (Dissertação de Mestrado) — Universidade Federal de Uberlândia, 2018.

PEIXOTO, D. C. F. **Geração Automática de Relatórios de Confiabilidade de Software: Projeto e Implementação de Engine para a Plataforma X-RAT**. Dissertação (Trabalho de Conclusão de Curso) — Universidade Federal de Uberlândia, Uberlândia, Brasil, janeiro 2023. Orientador: Rivalino Matias Jr.

PETRICEK, T. Computing and programming in context — introduction. In: **Proceedings of the Programming 2020**. [s.n.], 2020. Disponível em: <https://kar.kent.ac.uk/82634/>.

SALFNER, F.; LUTZ, P.; MALEK, M. **Fault Diagnosis in Computer Systems**. [S.l.]: Springer, 2006.

SANTOS, C. A. R. D.; MATIAS, R. J.; TRIVEDI, K. S. Abordagem estatística para predição de falhas de sistemas operacionais baseada em associação múltipla de falhas. **Revista Brasileira de Engenharia de Software**, v. 8, n. 2, p. 101–115, 2020.

SANTOS, C. A. R. dos; JR., R. M.; TRIVEDI, K. S. A multisite characterization study on failure causes in system and applications software. In: FEDERAL UNIVERSITY OF UBERLANDIA AND DUKE UNIVERSITY. **Proceedings of the Brazilian Symposium on Computing Systems Engineering (SBESC)**. [S.l.], 2021.

SMITH, L. Software failure analysis in modern operating systems. **Computer Systems Review**, v. 39, p. 67–81, 2023.

SOMMERVILLE, I. **Software Engineering**. 10th. ed. Harlow, England: Pearson, 2018.