ARGOS – Analisador Forense

Documento Plano de Testes

Histórico de Revisões

DATA	VERSÃO	DESCRIÇÃO	AUTOR
17/11/2025	1.0	Criação do documento e inserção das estrátegias	Nathalia G. Silva
19/11/2025	1.1	Criação da introdução e inserção de recursos	Kennedy R. T. Gonçalves
20/11/2025	1.2	Inserção dos riscos e níveis de teste	Kennedy R. T. Gonçalves
24/11/2025	1.3	Criação dos casos de teste	Nathalaia G. Silva
25/11/2025	1.4	Realização dos testes e criação dos relatórios (CT-01 até CT-08)	Kennedy R. T. Gonçalves e Nathalaia G. Silva
25/11/2025	1.5	Realização dos testes e criação dos relatórios (CT-09 até CT-16)	Kennedy R. T. Gonçalves e Nathalaia G. Silva

Sumário

1 INTRODUÇÃO

1.1 Propósito do documento

Este documento tem como propósito definir o planejamento, a estratégia e o controlo das atividades de teste para o projeto **Argos - Analisador Forense**. O objetivo principal é garantir que o software entregue cumpre os requisitos de segurança, desempenho e funcionalidade exigidos para uma ferramenta de monitorização de *logs* e resposta a incidentes em ambientes corporativos.

Este plano estabelece a abordagem para validar:

- **A Integridade da Solução:** Garantir que a arquitetura distribuída (microsserviços api_gestao, api_logs e analisador) funciona de forma coesa e resiliente em ambiente conterizado (Docker).

- **A Conformidade Funcional:** Verificar se as funcionalidades críticas, como a deteção heurística de atividades suspeitas, a gestão de perfis de acesso (RBAC) e a geração de relatórios de conformidade, operam conforme especificado nas Histórias de Usuário.

- **A Qualidade Não Funcional:** Validar requisitos críticos de desempenho (processamento em tempo real), segurança (proteção de dados e autenticação) e usabilidade da interface de monitorização.

Uma particularidade deste projeto é a sua natureza de **Software de Segurança Crítica**, o que exige um rigor adicional nos testes de carga e na validação de falsos positivos/negativos no motor de análise de ameaças, bem como a garantia de integridade imutável dos *logs* recolhidos para fins forenses.

1.2 Interessados

Aqui deverá constar uma lista dos interessados neste documento.

Perfil	Nome	Responsabilidade
Desenvolvedora / QA (Foco: Gestão e Compliance)	Kennedy R. T. Gonçalves	Responsável pela validação de todo o módulo de Gestão de Acessos (RBAC) e Autenticação. Cobrirá os testes de criação de utilizadores, perfis, bloqueio de permissões.
Desenvolvedor / QA (Foco:	Nathalia G. Silva	Responsável pela validação do

Monitoramento e Alertas)		núcleo de Segurança Operacional. Focará nos testes de ingestão de logs em tempo real, análise heurística, disparo de alertas e dashboards de visualização de ameaças.
Cliente (Simulado)	Equipa de SOC / DPO	Interessados nos resultados da validação (UAT) para garantir que o sistema deteta ameaças reais e protege a integridade dos dados forenses.

2 RECURSOS

2.1 Ambiente de testes

O ambiente de testes do projeto Argos é baseado numa arquitetura containerizada, garantindo a paridade entre os ambientes de desenvolvimento, teste e produção. A infraestrutura é orquestrada via **Docker Compose**, isolando os serviços e as suas dependências.

2.1.1 Hardware e Sistema Operativo (Host)

Para executar a bateria de testes (especialmente os Testes de Sistema e Carga), a máquina hospedeira (*Host*) deve cumprir os seguintes requisitos mínimos para suportar a execução simultânea de todos os contentores:

- **Sistema Operativo:** Windows 10/11 (com WSL2), Linux (Ubuntu 20.04+) ou macOS.
- **Processador:** CPU Multi-core (Intel i5/Ryzen 5 ou superior).
- **Memória RAM:** Mínimo de 8GB (Recomendado 16GB), devido ao consumo do Elasticsearch e MongoDB.
- **Armazenamento:** 20GB de espaço livre em disco para imagens Docker e volumes de logs.

2.1.2 Software e Ferramentas de Base

As seguintes ferramentas devem estar instaladas e configuradas na máquina do testador:

- **Docker Engine & Docker Compose:** Versão 24.0+ (ou Docker Desktop 4.20+).
- **Node.js:** Versão 18+ (Para execução de testes locais do Frontend).
- **Python:** Versão 3.11+ (Para execução de scripts de teste e automação local).
- **Git:** Para controlo de versão e obtenção do código-fonte.

2.1.3 Configuração dos Serviços (Ambiente Docker)

O ambiente de teste é composto pelos seguintes microsserviços, conforme definido na orquestração do projeto:

Serviço	Tecnologia / Versão	Porta Exposta	Função no Teste
api_gestao	Python 3.11 / FastAPI	8000	Validação de Autenticação, Perfis e Utilizadores.
api_logs	Python 3.12 / Flask	5000	Busca e Visualização de Logs em Tempo Real.
mongo	MongoDB 6.0	27017	Base de dados principal para gestão. Validação de persistência.
elasticsearch	Elastic 8.9.0	9200	Motor de busca de logs. Alvo de testes de carga.
filebeat	Elastic Filebeat 8.9.0	N/A	Agente de coleta. Validação do pipeline de ingestão.
gerador_logs	Python Script	N/A	Ferramenta de injeção de massa de dados para testes de stress.

2.1.4 Ferramentas de Apoio ao Teste

Para a execução e gestão dos casos de teste, serão utilizadas:

- **API Client:** Postman ou Insomnia (para testes manuais e exploratórios das APIs REST).
- **Navegadores:** Google Chrome (versão mais recente) e Mozilla Firefox (versão mais recente) para testes de compatibilidade do Frontend.
- **Frameworks de Teste:**
    - pytest (Backend): Para testes unitários e de integração de API.
    - Vitest (Frontend): Para testes de componentes React.
- **Scripts Auxiliares:**
    - seed.py: Script para popular a base de dados com massa de teste

inicial (Empresa, Gestor, Perfis).

- migrar_json.py: Script para validar migração de dados legados.

2.1.5 Dados de Teste

Os testes não devem ser executados com bases de dados vazias. O ambiente deve ser inicializado com o seguinte estado prévio:

- **Massa de Dados Mínima:** Execução obrigatória do script seed.py antes dos testes de validação para garantir a existência de um Gestor e Perfis padrão.
- **Logs Simulados:** O contentor gerador_logs deve estar ativo durante os testes de monitorização para prover um fluxo contínuo de dados.

2.1.6 Pessoas envolvidas

Descrever nesta seção as pessoas necessárias e o nível de envolvimento para execução do projeto de teste.

Nome	Setor	Contato
Kennedy R. T. Gonçalves	QA Lead / Gestão	krtorres.programador@hotmail.com
Nathalia G. Silva	QA Engineer / Automação	nathaliagoncalvessilva6@gmail.com

3 RISCOS

Esta secção identifica os potenciais obstáculos que podem impactar a execução dos testes ou a qualidade do produto, bem como as estratégias para os mitigar.

3.1 Mapeamento dos riscos

Risco	**Estratégia**	**Ação**
Sobrecarga da Equipa (Gargalo de Recursos) Risco de atraso nos testes devido à equipe ser composta apenas por 2 elementos acumulando desenvolvimento e QA.	Priorização rigorosa dos casos de teste baseada na criticidade das HUs (focar apenas nos "Deve Ter" do MoSCoW).	Reduzir o escopo dos testes de usabilidade (RNF013) e adiar a validação de funcionalidades de baixa prioridade (ex: relatórios secundários).
Instabilidade do Ambiente Docker Os contentores elasticsearch ou mongo podem falhar sob carga elevada durante os testes de sistema, bloqueando a execução.	Configuração de limites de recursos (mem_limit) no Docker e monitorização ativa durante a execução do gerador.py.	Reiniciar os serviços afetados via docker-compose restart e utilizar os *logs* de erro para isolar a causa raiz.
Indisponibilidade de Dependências Externas Falha na conexão com APIs de *Threat Intelligence* (ex: AbuseIPDB) durante os testes de integração, causando falsos negativos.	Utilização de *Mocks* (simuladores) nos testes automatizados para isolar o sistema da instabilidade da API externa.	Executar os testes em modo "offline" validando apenas a lógica interna de classificação de IPs.
Latência na Ingestão de Logs O *delay* entre a escrita do log pelo gerador.py e a indexação pelo Filebeat/Elasticsearch pode exceder o tempo esperado (5s), falhando os testes de tempo real.	Ajustar o intervalo de atualização (refresh_interval) do índice no Elasticsearch para o ambiente de testes.	Aumentar os *timeouts* nos scripts de teste automatizado e reportar como débito técnico de performance.
Dados de Teste Insuficientes O script gerador.py pode não cobrir todos os cenários de borda (ex: formatos de data inválidos ou *strings* corrompidas).	Revisão de código do gerador para incluir casos de "fuzzing" (dados aleatórios inválidos) nos logs gerados.	Inserção manual de logs "estragados" diretamente no volume partilhado para validar a robustez do *parser*.

4 ESTRATÉGIA DE TESTES

4.1 Teste de Unidade

Objetivo	Garantir que as menores unidades de código (funções, métodos e classes) funcionem corretamente de forma isolada, independente de dependências externas como bancos de dados (MongoDB/Elasticsearch) ou APIs de terceiros. O foco é na validação da lógica de negócio e tratamento de dados.
Abordagem Técnica	**Backend (Python):** Framework pytest para execução dos testes e unittest.mock para simular repositórios e chamadas externas. **Frontend (React/Vite):** Framework Vitest e React Testing Library para validação de componentes e *hooks*.
Cenário 1: **Backend - Núcleo de Segurança**	**O que testar:** • Verificar se a função gerar_hash_senha() retorna um hash válido e diferente da senha em texto plano. • Validar se verificar_senha() retorna True para senhas corretas e False para incorretas. • Testar a criação de tokens JWT, garantindo que o *payload* e o tempo de expiração estão corretos.
Cenário 2: **Backend - Serviços de Negócio**	**O que testar:** A lógica contida nas classes de serviço, isolando o acesso ao banco de dados (MongoDB) através de *Mocks*. • **AuthService:** Testar a lógica de autenticação (_autenticar_gestor/_autenticar_usuario), garantindo que o sistema rejeite usuários inativos ou senhas erradas antes mesmo de tentar gerar um token. • **UserService:** Testar o método criar_usuario para assegurar que ele gera corretamente o e-mail institucional e a senha aleatória antes de chamar o repositório. • **PermissaoService:** Testar a regra que impede a exclusão

	de uma permissão se ela estiver vinculada a um perfil, simulando o retorno do banco de dados.
Cenário 3: **Frontend - Lógica de Visualização**	**O que testar:** Os *Custom Hooks* que gerenciam o estado da aplicação. • **useLogDashboardViewModel:** Testar a função interna getSeverity(log), passando diferentes tipos de logs simulados (JSONs brutos) para garantir que a classificação de severidade ('WARNING', 'ERROR', 'INFO') segue as regras definidas. • **useAlertsViewModel:** Validar se a função getSeverity(motivo) mapeia corretamente strings como "ATIVIDADE SUSPEITA" para o nível "CRÍTICA".
Critérios de Aceite:	• Cada teste deve executar em menos de 100ms. • A cobertura de código (Code Coverage) para os módulos de services e core deve ser superior a 80%. • Todos os testes devem passar sem necessidade de levantar containers Docker ou conectar à rede.

4.1.1 Teste de Integração

Objetivo	Validar a interação entre os microsserviços do Argos (api_gestao, api_logs, analisador), a persistência de dados no MongoDB e Elasticsearch, e a comunicação assíncrona via logs partilhados (Filebeat). O foco é garantir que os dados fluem corretamente desde a ingestão até a visualização e alerta, sem perda de integridade ou falhas de autenticação.
Abordagem Técnica	Os testes serão executados num ambiente conteinerizado (Docker), replicando a infraestrutura de produção definida no docker-compose.yml.
Cenário 1:	**Contexto:** A api_gestao utiliza o ODM Beanie para gerir documentos no MongoDB. É crítico validar os "Links" (DBRefs) entre coleções.

Integração API de Gestão e MongoDB	**O que será testado:** • Criação de um Usuario (via UserController) vinculando-o a um Perfil Gestor existente. • Verificar se o campo fetch_links=True no UserRepository retorna corretamente o objeto Perfil aninhado no JSON de resposta, e não apenas o ID. **Critério de aceite:** O endpoint GET /usuarios/ deve devolver a estrutura completa do perfil (nome e permissões) associado ao utilizador, validando a integridade referencial do MongoDB.
Cenário 2: **Integração do Pipeline de Logs**	**Contexto:** O sistema depende do filebeat para ler o ficheiro /var/log/gerador.log partilhado pelo volume logs_volume e enviá-lo para o Elasticsearch. **O que será testado:** • O script gerador.py escreve uma entrada de log simulada no volume partilhado. Aguardar o ciclo de atualização do Filebeat. • Consultar a api_logs (rota /search) para confirmar se o log está indexado e disponível para busca. **Critério de aceite:** Um log gerado com um ID único deve aparecer na resposta JSON da api_logs em menos de 10 segundos.
Cenário 3: **Integração Analisador de Ameaças e Motor de Alertas**	**Contexto:** O serviço analisador consulta o Elasticsearch em loop à procura de padrões definidos em IPS_SUSPEITOS e escreve no índice alertas. **O que será testado:** • Injeção de um log contendo o IP suspeito 201.45.112.88 (Hardcoded no analisador.py) via gerador. • Verificar se o script analisador.py detecta este log e cria um documento no índice alertas. • Validar se o Frontend recebe este dado através da rota /alertas da api_logs. **Critério de aceite:** A injeção do IP suspeito deve resultar num

	novo item na lista de alertas da API sem intervenção manual.
Cenário 4: **Integração de Segurança**	**Contexto:** A autenticação é centralizada na api_gestao usando tokens JWT (HS256). **O que será testado:** • Login com credenciais de Gestor na rota /auth/login. • Utilização do token gerado para tentar aceder a uma rota protegida (/usuarios/). • Tentativa de acesso à mesma rota com um token expirado ou manipulado. **Critério de aceite:** O sistema deve permitir acesso com token válido (HTTP 200) e bloquear inequivocamente (HTTP 401/403) pedidos sem o cabeçalho Authorization correto, conforme definido nas dependências de segurança.

4.2 Teste de Validação

Objetivo	Confirmar se o sistema Argos atende aos Requisitos Funcionais e Histórias de Usuário (HUs) do MVP, definidos no Backlog do Produto, garantindo que as funcionalidades de monitorização, gestão de acesso e alertas entregam o valor esperado para as equipas de SOC e Compliance.
Abordagem Técnica:	Os testes serão realizados através da interface web (Frontend React), executando fluxos completos de ponta a ponta (E2E) que simulam a rotina diária de um analista de segurança.
Cenário 1: **Validação do Fluxo de Monitorização em Tempo Real (HU13)**	**Pré-condição:** O sistema estar recebendo logs do gerador.py e o utilizador estar logado com perfil de "Gestor". **Escopo:** • Abrir o menu "Monitoramento" (/logs). • Verificar se a tabela é atualizada automaticamente a cada 30

	segundos sem recarregar a página. • Utilizar a barra de pesquisa para filtrar por um IP específico (192.168.1.10). **Critério de aceite:** A tabela deve exibir apenas os logs correspondentes ao filtro e os contadores de estatística (Total, Alertas, Erros) devem refletir os dados filtrados.
Cenário 2: **Validação de Deteção e Alerta de Incidentes (HU16, HU17)**	**Pré-condição:** O serviço analisador estar ativo. **Escopo:** • Simular um ataque injetando um log com IP de origem 201.45.112.88 (conhecido como suspeito nas regras de negócio). • Aguardar o processamento (aprox. 10-30 segundos). • Navegar para o menu "Alertas" (/alerts). **Critério de aceite:** Deve aparecer um novo "Card de Alerta" na interface com severidade "CRÍTICA" ou "ALTA", indicando o motivo "Atividade suspeita detectada".
Cenário 3: **Validação de Gestão de Acessos e Perfis (HU1, HU7, HU8, HU12)**	**Pré-condição:** Estar logado como "Gestor". **Escopo:** • Abrir a menu "Gestão de Usuários" (/users). • Criar um "Perfil" chamado "Analista" selecionando apenas a permissão "Visualizar Logs". • Criar um utilizador associado a este perfil "Analista". • Fazer logout e login com o novo utilizador "Analista". • Tentar acessar à rota de "Gestão de Usuários". **Critério de aceite:** O sistema deve criar o perfil "Analista" e aparecer cadastrado em "Usuários Cadastrados". O sistema deve permitir o login, mas bloquear o acesso à página de gestão (redirecionar ou mostrar erro), pois o perfil "Auditor" não tem a permissão gerenciar_usuarios.
Cenário 4: **Validação de Autenticação e Sessão (HU3)**	**Escopo:** • Realizar login com credenciais válidas. • Fechar a aba do navegador e reabrir a aplicação.

	Critério de aceite: O utilizador deve permanecer logado ou ser capaz de restaurar a sessão via sessionStorage ou ser forçado a logar novamente de forma segura.

4.3 Teste de Sistema

Objetivo	Avaliar o comportamento do sistema Argos num ambiente de produção simulado, verificando o cumprimento dos Requisitos Não Funcionais (RNFs) de desempenho, segurança, confiabilidade e portabilidade definidos no Documento de Visão.
Abordagem Técnica:	O ambiente deve ser uma réplica exata da produção, utilizando a orquestração definida no docker-compose.yml, com todos os microsserviços (api_gestao, api_logs, analisador, elasticsearch, mongo, filebeat) ativos e comunicantes.
Cenário 1: **Teste de Carga e Desempenho (Teste de Estresse)**	**Requisito:** RNF002 (Processamento em Tempo Real < 5s para 1M registos) e RNF016 (Escalabilidade). **Escopo:** • Configurar o script gerador.py para emitir logs em alta frequência (reduzir o time.sleep para 0.01s). • Monitorizar o consumo de memória e CPU dos contêiner api_logs e elasticsearch. • Medir a latência entre a escrita no gerador.log e a disponibilidade na rota /search da API. **Critério de Aceite:** O sistema não deve bloquear (crash) e a latência de ingestão deve manter-se estável mesmo sob carga elevada.
Cenário 2: **Teste de Recuperação de Falhas**	**Requisito:** RNF009 (Tolerância a Falhas e Recuperação) e RNF008 (Disponibilidade). **Escopo:** • Com o sistema em execução, forçar a paragem do contêiner do banco de dados (docker stop mongo). • Verificar se as APIs (api_gestao e api_logs) tratam o erro adequadamente retornando mensagens de erro amigáveis ou códigos 503, sem expor *stack traces*. • Reiniciar o contêiner parado.

	• Verificar se os microsserviços reconectam automaticamente sem necessidade de reinício manual. **Critério de Aceite:** O sistema deve recuperar a funcionalidade total automaticamente após o restabelecimento do serviço dependente.
Cenário 3: **Teste de Segurança (Scanner de Vunerabilidade)**	**Requisito:** RNF018 (Validação de Entrada - OWASP) e RNF003 (Proteção de Dados). **Escopo:** • Executar ferramentas de análise dinâmica (DAST) como OWASP ZAP contra as rotas da API (/auth/login, /search). • Tentar injeção de NoSQL nos campos de filtro da api_logs (enviar { $ne: null } nos parâmetros de busca). • Verificar se o *token* JWT no sessionStorage do Frontend não contém dados sensíveis desnecessários no *payload*. **Critério de Aceite:** Nenhuma vulnerabilidade de severidade Alta ou Crítica deve ser detectada. O sistema deve remover todas as entradas de pesquisa.

4.4 Técnicas de teste

Para garantir a cobertura completa dos requisitos funcionais e não funcionais do sistema Argos, será utilizada uma abordagem híbrida combinando técnicas de Caixa Branca (focada no código) e Caixa Preta (focada no comportamento), aplicadas através de execuções manuais e automatizadas.

4.5 Testes Manuais

A execução manual será prioritária para validar a experiência do utilizador e fluxos complexos que requerem julgamento humano.

Teste Exploratório	Os analistas de QA navegarão livremente pela aplicação sem guiões rígidos para identificar comportamentos imprevistos, especialmente nos Dashboards de Monitorização e na visualização de gráficos. O objetivo é simular o uso real de um gestor de SOC sob pressão.

Teste de Usabilidade (Verificação de Interface)	Validação visual do cumprimento das regras de CSS/Layout (Tailwind) e responsividade. Verificação da clareza das mensagens de erro e *feedback* visual (ex: *Loaders* e *Toasts* de sucesso/erro) definidos nos *ViewModels*.
Avaliação Heurística de Nielsen	Inspeção visual das interfaces principais (Dashboard de Logs, Alertas e Gestão de Usuários) focada na conformidade com três heurísticas críticas para o projeto Argos: **Visibilidade do Estado do Sistema:** O sistema deve manter o utilizador sempre informado sobre o que está a acontecer. Aplicação no Argos*:* Verificar se o utilizador recebe *feedback* visual imediato (*spinners* ou barras de progresso) enquanto a aplicação aguarda a resposta da api_logs ao filtrar grandes volumes de dados ou se a conexão com o Elasticsearch cair. **Correspondência entre o Sistema e o Mundo Real:** O sistema deve falar a linguagem do utilizador (termos de cibersegurança), e não a do sistema (termos de programação). Aplicação no Argos*:* Garantir que os Alertas utilizam terminologia adequada a um SOC (ex: "IP Suspeito", "Força Bruta", "Severidade Crítica") em vez de códigos de erro técnicos ou nomes de variáveis internas (ex: "Error 500", "Null Pointer"). **Prevenção de Erros:** O sistema deve ser desenhado para evitar que problemas ocorram, em vez de apenas mostrar mensagens de erro. Aplicação no Argos: Validar se o formulário de "Novo Usuário" impede proativamente o cadastro com dados inválidos (desabilitar o botão "Salvar" se o e-mail estiver mal formatado ou se o perfil não for selecionado) antes mesmo de enviar a requisição ao servidor.
Validação de Instalação	Execução manual dos passos descritos no README.md e docker-compose.yml para garantir que o ambiente sobe corretamente numa infraestrutura limpa.

4.6 Testes Automatizados

A automação será a base da estratégia de regressão e validação contínua, integrada no ciclo de *build*.

Testes de Unidade (Caixa Branca)	Utilização de pytest (Backend) e Vitest (Frontend) para testar métodos e funções isoladas. O foco será na lógica de negócio crítica, como o cálculo de severidade de logs e validação de *tokens* JWT.

Testes de API (Integração)	*Scripts* automatizados que disparam requisições contra os *endpoints* da api_gestao e api_logs para validar códigos de estado HTTP (200, 401, 403, 500) e a estrutura dos JSONs de resposta.
Testes de Carga (Scripted)	Utilização do script gerador.py para injeção massiva de dados simulados, permitindo validar a estabilidade do sistema sob alto volume de processamento sem intervenção humana.

5 Técnicas Específicas de Design de Teste

Para a elaboração dos casos de teste, serão aplicadas as seguintes técnicas formais:

- **Particionamento de Equivalência:** Aplicado nos formulários de cadastro.

  Exemplo: Se o campo "Nome" aceita strings não vazias, testaremos apenas um caso válido ("João") e um inválido ("") para cobrir a regra, reduzindo o número total de testes necessários.

- **Análise de Valor Limite:** Focada nos parâmetros de pesquisa de logs e datas.

  Exemplo: Testar pesquisas com 0 resultados, 1 resultado e o limite máximo de paginação (50 logs) para garantir que a API lida corretamente com os extremos.

- **Injeção de Falhas:** Técnica utilizada nos Testes de Sistema para simular a queda de serviços dependentes (ex: parar o contentor mongo ou elasticsearch) e verificar se a aplicação trata a exceção graciosamente sem expor dados sensíveis.

5.1 Cobertura de código

A medição da cobertura de código será utilizada para quantificar a eficácia dos testes unitários e de integração, garantindo que as áreas críticas do sistema Argos (especialmente lógica de segurança e tratamento de dados) foram exercitadas.

5.1.1 Ferramentas de Medição

Serão utilizadas ferramentas nativas e extensões padrão da indústria para cada parte da arquitetura:

- **Backend (Python - api_gestao e api_logs):**

Ferramenta	pytest-cov (plugin do pytest que utiliza Coverage.py).
Método	Execução automatizada durante o pipeline de *build*. O relatório deve incluir a cobertura de linhas (line coverage) e de ramificação (branch coverage) para garantir que todos os caminhos condicionais (if/else)

	foram testados.
Alvo Principal	Diretórios app/services/ e app/core/, onde reside a lógica de autenticação e regras de negócio.

- **Frontend (React - argos/frontend):**

Ferramenta	Vitest com o provedor de cobertura nativo @vitest/coverage-v8 (padrão para projetos Vite).
Método	Execução de *scripts* de teste (npm run coverage) focados na lógica de apresentação e gestão de estado.
Alvo Principal	Diretório src/ViewModels/[3], onde estão isoladas as regras de interface e chamadas de API.

5.1.2 Metas de Cobertura

Considerando a natureza crítica do software (Ferramenta Forense e de Segurança), as metas estabelecidas são:

Nível	Percentual	Ação em caso de falha
Necessário (Mínimo)	60%	Bloqueio do *Merge Request* / Falha no Pipeline de CI.
Desejável (Meta)	80%	Aprovação automática e selo de qualidade "A".

5.1.3 Políticas de Exclusão

Para evitar distorções nas métricas, os seguintes componentes serão excluídos da contagem obrigatória de cobertura:

- Ficheiros de configuração (config.py, vite.config.js).
- Scripts de migração de banco de dados e *seeds* (migrar_json.py, seed.py).
- Interface do Utilizador (UI) pura sem lógica (apenas renderização de CSS/HTML).
- *Wrappers* de bibliotecas de terceiros (código boilerplate do Elasticsearch e Beanie).

6 NÍVEIS DE TESTES

Nesta secção, estão definidos os tipos de teste específicos que serão executados para validar cada dimensão de qualidade do sistema Argos, garantindo a conformidade com os Requisitos Não Funcionais (RNFs).

Dimensão de Qualidade/ Risco de Qualidade	**Tipo de Teste**
Funcionalidade	**Teste de função:** Validação de todas as Histórias de Usuário (HUs) implementadas, garantindo que os fluxos de Gestão de Acesso (API Gestão) e Monitorização de Logs (API Logs) produzem os resultados esperados. **Foco:** Login, CRUD de Utilizadores, Visualização de Dashboards e Receção de Alertas. **Teste de segurança:** Verificação crítica dos mecanismos de proteção. Inclui: **Autenticação:** Teste de robustez do JWT e expiração de sessões. **Controlo de Acesso (RBAC):** Garantir que perfis sem permissão não acedem a rotas administrativas. **Teste de volume:** Verificação da estabilidade do banco de dados MongoDB e do Elasticsearch quando populados com o volume de dados histórico esperado (1 ano/10TB conforme RNF015), garantindo que as consultas não excedem 30s (RNF006).
Utilidade	**Teste de usabilidade:** Validação da consistência visual (CSS/Tailwind) e da responsividade dos Dashboards em diferentes resoluções de ecrã. **Foco:** Verificar se os gráficos e tabelas de logs se adaptam corretamente e se o *feedback* visual (spinners, toasts) é claro para o analista.
Confiabilidade	**Teste de Recuperação:** Simulação de falhas na infraestrutura Docker (ex: paragem abrupta do contentor mongo ou filebeat) para validar se o sistema recupera a operação sem perda de integridade dos dados (RNF009).
Desempenho	**Teste de carga:** Avaliação da capacidade do sistema em processar grandes volumes de dados em tempo real (RNF002). **Método:** Utilização do script gerador.py em múltiplas instâncias para simular a ingestão de milhares de logs por segundo e medir a latência de indexação no Elasticsearch.
Suportabilidade	**Teste de instalação:** Validação da portabilidade da infraestrutura contentorizada. O comando docker-compose up deve funcionar consistentemente em ambientes Linux (Ubuntu) e Windows (WSL2), garantindo a independência do Sistema Operativo hospedeiro (RNF012).

7 CASOS DE TESTES

Nesta secção são detalhados os casos de teste prioritários para a validação da versão 2.0.0 do sistema Argos. Estes casos cobrem os fluxos críticos de segurança e operação.

7.1 Módulo de Gestão e Segurança (API Gestão)

7.1.1 Teste de login

ID do Teste	CT-01: Login de Gestor e Emissão de Token
Rastreabilidade	RNF003 (Segurança), HU1 e HU12
Estratégia	5.1.2 – (Cenário)
Pré-condição	O utilizador "Gestor" deve estar criado na base de dados (via seed.py) e a api_gestao ativa.
Passos Executados	1. Aberta à página de Login. 2. Inserido e-mail: gestor@argos.com. 3. Inserido senha: senha123. 4. Selecionado "Entrar". 5. Verificado o SessionStorage do navegador.
Resultado Esperado	O sistema deve redirecionar para o Dashboard. Um token JWT válido deve estar armazenado com a chave access_token.
Resultado Obtido	O sistema redirecionou para o dashboard. Um token JWT válido foi armazenado com a chave de acesso.
Evidências	Anexo: Figura 1 - Token JWT válido
Status	Aprovado

7.1.2 RBAC

ID do Teste	CT-02: Bloqueio de Acesso por Perfil (RBAC)
Rastreabilidade	HU12 e HU7
Estratégia	5.1.3 - Cenário 3
Pré-condição	Estar logado com um utilizador de perfil "Analista" (que não possui a permissão gerenciar_usuarios).
Passos Executados	1. Tentativa em abrir diretamente a URL /users no navegador. 2. Observado o comportamento da interface. 3. Tentativa em enviar um POST via Postman para

	http://localhost:8000/usuarios/ usando o token deste analista.
Resultado Esperado	Interface: Deve redirecionar para a página de analista (Monitoramento e Alertas), impedindo a visualização da Gestão de Usuário. API: Deve retornar HTTP 403 Forbidden.
Resultado Obtido	Interface: O usuário foi redirecionado para a página de gestor e não para a de analista. API: Retorna HTTP 403 Forbidden.
Evidências	Anexo: Figura 2 – Conta de usuário teve o acesso negado. Figura 3 – Usuário não possui permissão para acessar "/users".
Status	Defeito na Interface / API Aprovada

7.2 Monitoramento e Alertas

7.2.1 Monitoramento em tempo real

ID do Teste	CT-03: Fluxo de Ingestão em Tempo Real
Rastreabilidade	HU13 e RNF002
Estratégia	5.1.3 – Cenário 1
Pré-condição	Contêiner gerador_logs, filebeat e elasticsearch ativos.
Passos Executados	1. Aberto o Dashboard de Logs. 2. Confirmado a hora (22:28:55) do último log visível na tabela. 3. Aguardado 30 segundos sem interagir com a página. 4. Verificado se novos registos apareceram no topo da tabela.
Resultado Esperado	A tabela deve atualizar automaticamente (via *polling* do useRealTimeLogViewModel), exibindo logs gerados há menos de 1 minuto.
Resultado Obtido	O dashboard de logs foi atualizado automaticamente em menos de 1 minuto.
Evidências	Anexo: Figura 4 – Dashboard de logs e Figura 5 – Dashboard após 30 segundos.
Status	Aprovado

7.2.2 Detecção de Ameça

ID do Teste	CT-04: Deteção de Ameaça (IP Suspeito)
Rastreabilidade	HU16 e HU17
Estratégia	5.1.2 – Cenário 3 e 5.1.2 – Cenário 2
Pré-condição	O serviço analisador deve estar em execução.
Passos Executados	1. Foi forçado a geração de um log de ataque executando manualmente no terminal: docker exec -it gerador_logs python -c "import gerador; f=open('/var/log/gerador.log','a'); gerador.gerar_log_falha_multipla(f); f.close()". 2. Aguardado 1 minuto. 3. Aberto a aba "Alertas" no Frontend.
Resultado Esperado	Deve surgir um novo card de alerta com severidade "CRÍTICA" ou "ALTA", indicando "Atividade suspeita detectada do IP: 201.45.112.88".
Resultado Obtido	Apareceu um novo card de alerta com severidade “CRÍTICA” para o IP: 201.45.112.88.
Evidências	Anexo: Figura 6 – Detectada novo card de alerta
Status	Aprovado

7.3 Testes de Unidade

Este caso de teste valida a lógica interna de segurança, sem depender da base de dados ou API.

7.3.1 Validação de Hash de Senha

ID do Teste	CT-05: Validação de Hash de Senha
Rastreabilidade	HU1
Estratégia	5.1.1 – Cenário 1
Pré-condição	Ambiente de desenvolvimento Python configurado com pytest instalado.
Passos Executados	1. Executado o script de teste unitário na raiz argos/: docker-compose exec api_gestao python -m pytest tests/unit/test_security.py -v 2. O teste solicitou a função gerar_hash_senha ("senha123"). 3. O teste solicitou verificar_senha ("senha123", hash_gerado) e verificar_senha ("senhaErrada", hash_gerado).
Resultado Esperado	O hash gerado não deve ser igual à string "senha123". A verificação da senha correta deve retornar True.

	A verificação da senha errada deve retornar False. O teste deve passar em menos de 100ms.
Resultado Obtido	test_funcionalidade_hash_senha **PASSED**
Evidências	Anexo: Figura 7 – Teste de validação de hash
Status	Aprovado

7.3.2 Criação de Usuário

ID do Teste	CT-06: Criação de Usuario
Rastreabilidade	HU1
Estrátegia	5.1.1 – Cenário 2
Pré-condição	Ambiente Python com pytest e unittest.mock.
Passos Executados	1. Executado o teste unitário para o UserService utilizando na raiz “argos/”: docker-compose exec api_gestao python -m pytest tests/unit/test_user_service.py -v 2. O teste simulou o método user_repository.create para não bater no banco. 3. Solicitado criar_usuario passando o nome "João Silva". 4. Verificado se o e-mail gerado internamente foi joao.silva@argos.com antes de tentar salvar.
Resultado Esperado	O e-mail institucional deve ser gerado corretamente (nome.sobrenome@dominio). A função deve chamar o repositório exatamente uma vez. O teste deve passar isoladamente sem requerer o MongoDB ativo.
Resultado Obtido	test_criar_usuario_gera_email_correto **PASSED**
Evidências	Anexo: Figura 8 – Criação de usuário e geração de email institucional
Status	Aprovado

7.3.3 Severidade de Logs

ID do Teste	CT-07: Lógica de Severidade de Logs
Rastreabilidade	HU13
Estratégia	5.1.1 – Cenário 3
Pré-condição	Ambiente Node.js com Vitest instalado.
Passos Executados	1. Executado o teste unitário: npm test src/ViewModels/useLogDashboardViewModel.test.js. 2. O teste importou a função getSeverity isolada. 3. Passou um log com categoria: "suspeito" e verificou o retorno. 4. Passou um log com mensagem: "Login efetuado" e verificou o retorno.
Resultado Esperado	Para a categoria "suspeito", a função deve retornar string 'WARNING'. Para a mensagem normal, deve retornar 'INFO'. Isto garante que os alertas visuais aparecerão corretamente para o utilizador.
Resultado Obtido	✓ deve retornar WARNING quando a categoria for "suspeito" 1ms ✓ deve retornar INFO para mensagem "Login efetuado" (sem indícios de erro) 0ms ✓ deve retornar ERROR quando a mensagem contém "FALHA" 0ms **3 passed**
Evidências	Anexo: Figura 09 – Prompt de retornos de severidade
Status	Aprovado

7.4 Testes de Integração

7.4.1 Integridade do Banco de Dados

ID do Teste	CT-08: Integridade Referencial Usuário-Perfil
Rastreabilidade - Estratégia	5.1.2 - Cenário 1
Pré-condição	api_gestao e mongo ativos. Criar usuário
Passos Executados	1. Realizado uma requisição GET /usuarios/ via Postman (usando token de Gestor). 2. Analisado o JSON de resposta para o utilizador "Sherlock".

	3. Verificado o campo perfil.
Resultado Esperado	O campo perfil deve ser um objeto contendo "nome" e "permissoes", confirmando que o fetch_links=True funcionou no Backend.
Resultado Obtido	Perfil armazena
Evidências	Anexo: Figura 10 – Requisição GET para atribuição de permissão ao respectivo perfil e Figura 11 – Evidência das respectivas permissões relacionadas ao perfil criado.
Status	Aprovado

7.4.2 Integração do Pipeline de Ingestão

ID do Teste	CT-09: Integração do Pipeline de Ingestão
Rastreabilidade - Estratégia	5.1.2 - Cenário 2: Pipeline de Logs
Pré-condição	Contêiner filebeat e elasticsearch ativos. Acesso ao terminal.
Passos Executados	1. Identificado o ID do contêiner do gerador: `docker ps
Resultado Esperado	A API deve retornar um JSON contendo o log injetado. O campo mensagem deve ser exatamente "Log Marcado Único 12345". Isto confirma que o Filebeat leu o ficheiro e o Elasticsearch indexou o documento corretamente.
Resultado Obtido	O filebeat não conseguiu processar o arquivo dessa forma o log não chegou ao elasticsearch.
Evidências	Anexo : Figura 12 – Erro de Integração conteúdo vázio ‘[ ]’
Status	Defeito

7.5 Teste de Validação

Este cenário valida o fluxo completo de gestão.

7.6 Validação de Gestão de Acessos e Perfis

ID do Teste	CT-10: Ciclo de Vida de Gestão de Usuários
Rastreabilidade	HU1, HU7, HU8 e HU12
Estratégia	5.1.3 – Cenário 3
Pré-condição	Logado como "Gestor" na interface Web.
Passos Executados	1. Selecionado "Novo Usuário" e preenchido: Nome "Teste QA", Email "qa@argos.com", Perfil "Analista". 2. Confirmado que o usuário aparece na tabela com

	status "ATIVO". 3. Selecionado no botão de "Desativar" (ícone de bloqueio). 4. Verificado se o status mudou visualmente para "DESATIVADO".
Resultado Esperado	Usuário irá permanecer com o respectivo status até que seja alterado
Resultado Obtido	"POST /usuarios/…/desativar HTTP/1.1" 200 OK "POST /usuarios/…/ativar HTTP/1.1" 200 OK
Evidências	Anexo: Figura 13 – Rota integra
Status	Aprovado

7.7 Testes de Sistema

Este teste válido a resiliência da arquitetura Docker, simulando uma falha crítica em produção.

7.7.1 Recuperação Automática de Falha

ID do Teste	CT-11: Recuperação Automática de Falha no Banco
Rastreabilidade	RNF009 e RNF008
Estratégia	5.1.4 – Cenário 2
Pré-condição	Sistema em execução e api_gestao respondendo.
Passos Executados	1. No terminal, executando: docker stop mongo. 2. Tentativa de fazer Login na aplicação (deve falhar/dar erro 500). 3. Executando: docker start mongo. 4. Aguardando 10 segundos. 5. Tentativa de fazer Login novamente.
Resultado Esperado	O sistema deve falhar de forma controlada. O sistema deve voltar a permitir o login automaticamente, sem necessidade de reiniciar o contêiner da api_gestao.
Resultado Obtido	Code 200 **Successful Response**
Evidências	Anexo: Figura 14 – Falha induzida no mongo
Status	Aprovado

7.8 Teste de Stress e Carga

ID do Teste	CT-12: Teste de Stress e Carga
Rastreabilidade - Estratégia	5.1.4 - Cenário 1
Pré-condição	Ambiente Docker ativo. Ferramenta de monitorização (docker stats) aberta.
Passos Executados	1. Alterado o docker-compose.yml para escalar o gerador: deploy: replicas: 5 ou correr 5 instâncias manuais. 2. Executado docker-compose up -d --scale gerador_logs=5. 3. Monitorizado o consumo de RAM do contentor elasticsearch durante 10 minutos. 4. Realizado pesquisas na API de Logs simultaneamente.
Resultado Esperado	O contentor do Elasticsearch não deve reiniciar (Crash) por falta de memória. A API de Logs deve continuar a responder (HTTP 200) em menos de 5 segundos, mesmo sob a carga de 5 geradores.
Resultado Obtido	Elasticsearch não caiu e a API continuou respondendo
Evidências	Anexo: Figura 15 - Performance da máquina Figura 16 – 5 instâncias de gerador = x5 velocidade de gravação de logs
Status	Aprovado

7.8.1 Segurança Ofensiva

ID do Teste	CT-13: Segurança - Tentativa de Injeção NoSQL
Rastreabilidade - Estratégia	5.1.4 - Cenário 3
Pré-condição	api_logs ativa e acessível em http://localhost:5000.
Passos Executados	1. Utilizado o navegador. 2. Enviado um pedido GET malicioso para tentar listar tudo sem filtro: GET http://localhost:5000/search?q={"$ne": null} (tentativa de injetar operador MongoDB/Elastic). 3. Enviado caracteres especiais de quebra de query: GET http://localhost:5000/search?q=' OR 1=1.
Resultado Esperado	A API deve bloquear a entrada e tratar os caracteres como texto literal.

	O sistema não deve retornar erros de *stack trace* do Python/Flask expostos ao utilizador. Deve retornar 0 resultados ou apenas logs que contenham literalmente aquele texto.
Resultado Obtido	O sistema retorna 0 resultados.
Evidências	Anexo: Figura 17 – Print retorno de pagina não encontrada.
Status	Aprovado

7.9 Testes Manuais e Usabilidade

Este caso de teste formaliza a inspeção técnica da interface, garantindo que os princípios de Nielsen definidos na estratégia são verificados.

7.9.1 Heurísticas de Nielsen

ID do Teste	CT-14: Avaliação Heurística da Interface
Rastreabilidade - Estrátegia	5.2.1
Heurísticas	Visibilidade e Prevenção de Erros
Pré-condição	Acesso ao Frontend com perfil de "Gestor".
Passos Executados	1. Visibilidade: Forçado um carregamento lento (ex: *throttling* de rede no F12) aos filtrados logs e verificado se aparece um indicador visual (*spinner*). 2. Prevenção: Abrindo o formulário de "Novo Usuário" e tentado clicar em "Salvar" com o campo e-mail vazio. O botão deve estar desabilitado ou não reagir. 3. Linguagem: Verificado se os cards de alerta usam termos como "IP Suspeito" em vez de códigos de erro técnicos.
Resultado Esperado	O sistema deve fornecer *feedback* visual imediato em todas as ações demoradas. O formulário deve bloquear o envio de dados inválidos antes da requisição. A terminologia deve ser adequada ao domínio de segurança.
Resultado Obtido	O sistema fornece o feedback visual, bloqueia o envio de dados inválidos e se adequa a terminologia de segurança.
	Anexo: Figura 18 – Visibilidade: Forçando um carregamento lento; Figura 19: Prevenção: Indicação de campo vazio; Figura 20: Terminologia no painel de monitoramento
Status	Aprovado

7.9.2 Responsividade e Compatibilidade Visual

ID do Teste	CT-15: Responsividade e Compatibilidade Visual
Rastreabilidade - Estratégia	5.2.1 e RNF020
Pré-condição	Acesso ao sistema via Chrome, Firefox e Edge.
Passos Executados	1. Aberto o Dashboard de Logs no navegador *chrome*. 2. Reduzidos a janela do navegador para simular um ecrã de *tablet* (largura 768px). 3. Verificado se a tabela de logs se adapta (barra de rolagem horizontal ou empilhamento) sem quebrar o *layout*.
Resultado Esperado	A interface deve manter a consistência visual (cores, fontes, alinhamento) em todos os navegadores. Não devem existir elementos sobrepostos ou ilegíveis em resoluções menores.
Resultado Obtido	A interface mantem a consistência visual em todos os navegadores, sem elementos sobre postos ou ilegíveis.
Evidências	Anexo: Figura 21 – Ecrã de tablet (largura 768px)
Status	Aprovado

7.10 Testes Manuais

7.10.1 Sessão de Teste Exploratório

ID do Teste	CT-16: Sessão de Teste Exploratório
Rastreabilidade - Estratégia	5.2.1
Pré-condição	Sistema com dados populados (seed.py + gerador.py).
Missão (Objetivo)	Investigar o comportamento dos gráficos e filtros de data quando submetidos a intervalos de tempo inválidos ou sem dados.
Execução	O testador deve navegar livremente durante 30 minutos, tentando "partir" a visualização.
Resultado Esperado	O sistema não deve apresentar ecrãs brancos ou erros de JavaScript não tratados na consola. Deve exibir mensagens como "Sem dados neste período".
Resultado Obtido	O sistema atende aos requisitos.
Evidências	Anexo: Figura 22 – Mensagem de nenhum usuário encontrado
Status	Aprovado

8 Anexos

Caso de teste 1 (CT-01) - Teste de login

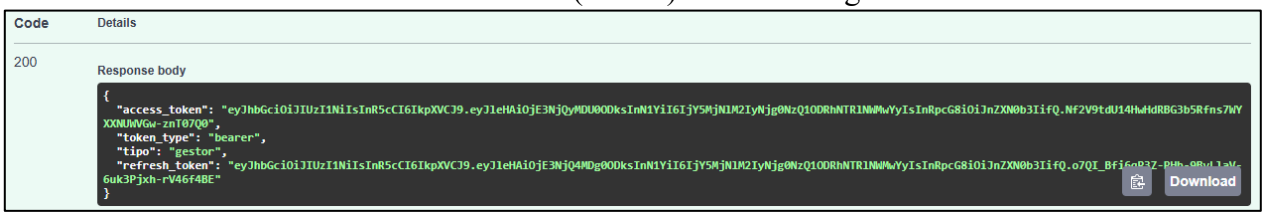

Figura 2 - Token JWT válido
Fonte: Elaborada pelo autor (2025)

Caso de teste 2 (CT-02) – RBAC

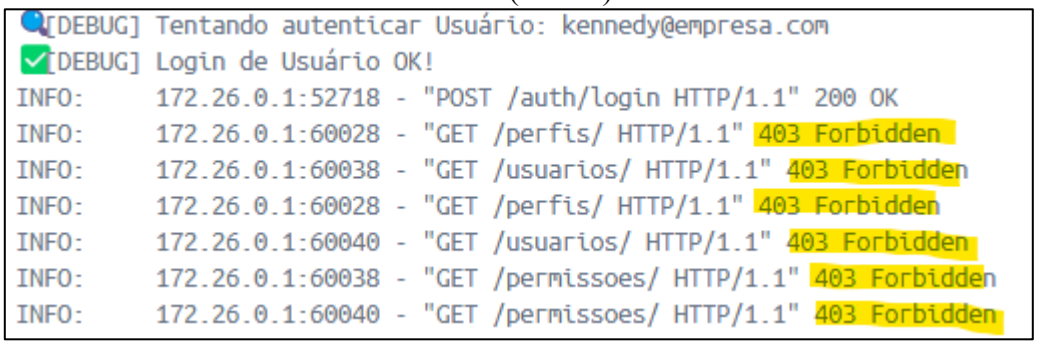

Figura 2 – RBAC – Conta de usuário teve o acesso negado
Fonte: Elaboração própria (2025)

Caso de teste 2 (CT-02) – RBAC

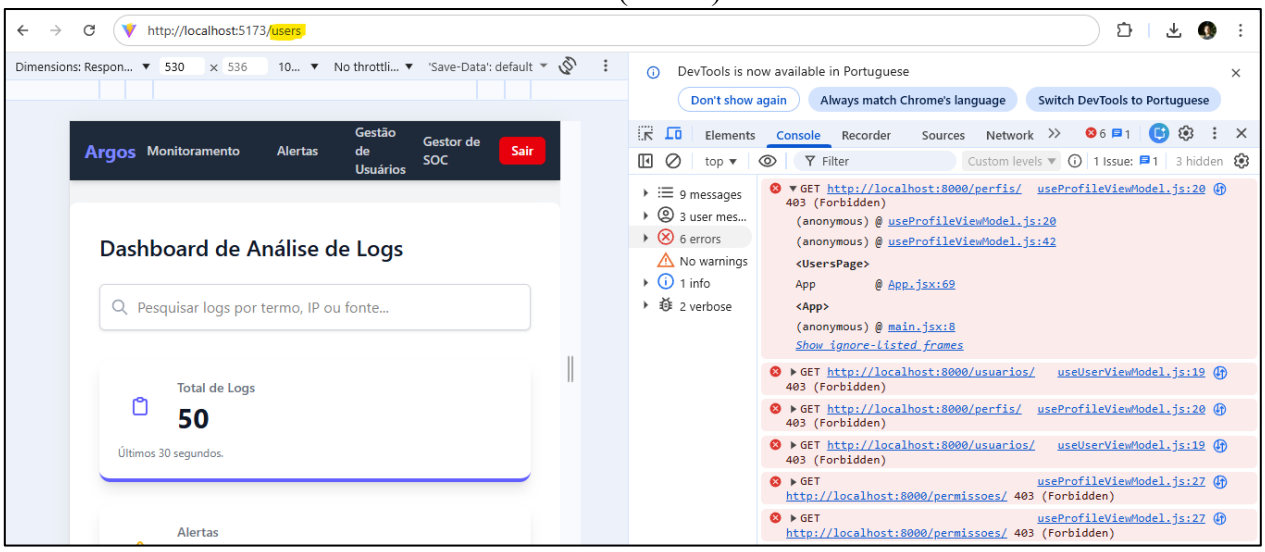

Figura 3 – RBAC – Usuário não possui permissão para acessar "/users"
Fonte: Elaboração própria (2025)

Caso de teste 3 (CT-03) - Monitoramento em tempo real

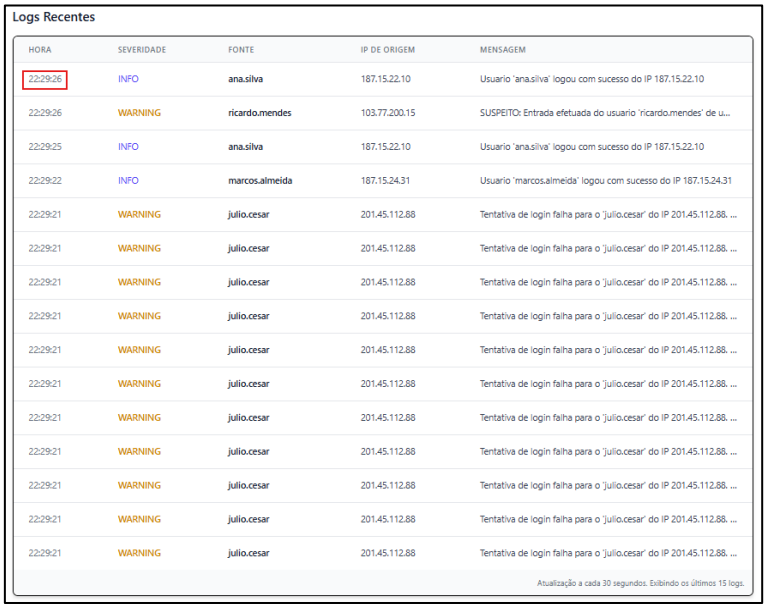

Figura 4 – dashboard de logs
Fonte: Elaboração própria (2025)

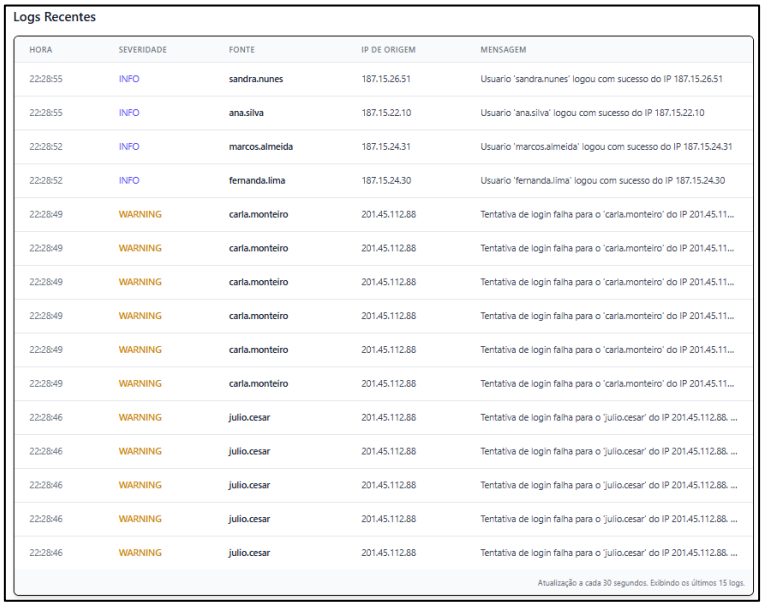

Figura 5 – dashboard após 30 segundos
Fonte: Elaboração própria (2025)

Caso de teste 4 (CT-04) - Deteção de Ameaça (IP Suspeito)

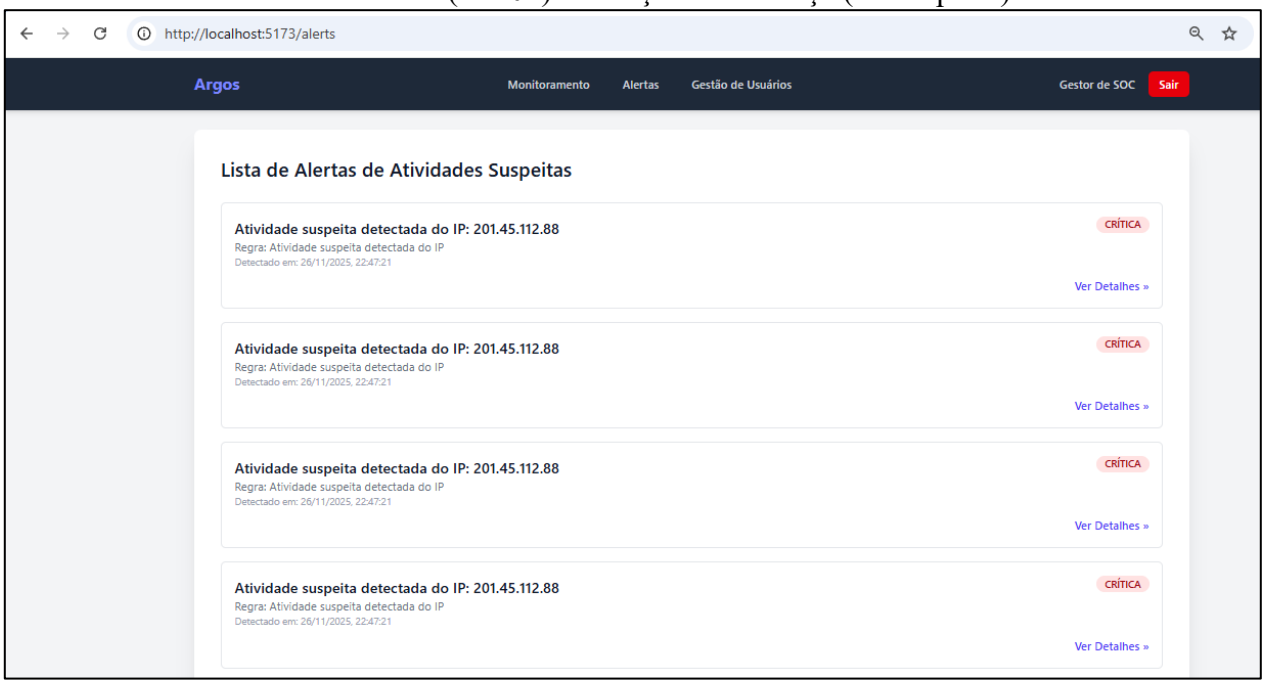

Figura 6 – Detectado novo card de alerta
Fonte: Elaboração própria (2025)

Caso de teste 5 (CT-05) - Validação de Hashing de Senha

```
(.venv) PS D:\OneDrive\Documentos\GitHub\SolucoesComputacionais-ArgosForense\Analisador-Forense\argos> docker-compose ex
ec api_gestao python -m pytest tests/unit/test_security.py -v
time="2025-11-27T01:23:17-03:00" level=warning msg="D:\\OneDrive\\Documentos\\GitHub\\SolucoesComputacionais-ArgosForens
e\\Analisador-Forense\\argos\\docker-compose.yml: `version` is obsolete"
============================================= test session starts =============================================
platform linux -- Python 3.11.14, pytest-7.4.3, pluggy-1.6.0 -- /usr/local/bin/python
cachedir: .pytest_cache
rootdir: /app
plugins: anyio-4.11.0
collected 1 item

tests/unit/test_security.py::test_funcionalidade_hash_senha PASSED                                       [100%]

============================================== warnings summary ===============================================
../usr/local/lib/python3.11/site-packages/passlib/utils/__init__.py:854
  /usr/local/lib/python3.11/site-packages/passlib/utils/__init__.py:854: DeprecationWarning: 'crypt' is deprecated and s
lated for removal in Python 3.13
    from crypt import crypt as _crypt

../usr/local/lib/python3.11/site-packages/pydantic/_internal/_config.py:323
  /usr/local/lib/python3.11/site-packages/pydantic/_internal/_config.py:323: PydanticDeprecatedSince20: Support for clas
s-based `config` is deprecated, use ConfigDict instead. Deprecated in Pydantic V2.0 to be removed in V3.0. See Pydantic
V2 Migration Guide at https://errors.pydantic.dev/2.11/migration/
    warnings.warn(DEPRECATION_MESSAGE, DeprecationWarning)

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
========================================== 1 passed, 2 warnings in 0.93s ==========================================
```

Figura 7 – Validação de hash
Fonte: Elaboração própria (2025)

Caso de teste 6 (CT-06) - Criação de Usuario

```
(.venv) PS D:\OneDrive\Documentos\GitHub\SolucoesComputacionais-ArgosForense\Analisador-Forense\argos> docker-compose ex
ec api_gestao python -m pytest tests/unit/test_user_service.py -v
time="2025-11-27T02:00:11-03:00" level=warning msg="D:\\OneDrive\\Documentos\\GitHub\\SolucoesComputacionais-ArgosForens
e\\Analisador-Forense\\argos\\docker-compose.yml: `version` is obsolete"
============================================= test session starts =============================================
platform linux -- Python 3.11.14, pytest-7.4.3, pluggy-1.6.0 -- /usr/local/bin/python
cachedir: .pytest_cache
rootdir: /app
plugins: asyncio-0.23.5, anyio-4.11.0
asyncio: mode=Mode.STRICT
collected 1 item

tests/unit/test_user_service.py::test_criar_usuario_gera_email_correto PASSED                            [100%]

============================================== warnings summary ===============================================
../usr/local/lib/python3.11/site-packages/pydantic/_internal/_config.py:323
../usr/local/lib/python3.11/site-packages/pydantic/_internal/_config.py:323
../usr/local/lib/python3.11/site-packages/pydantic/_internal/_config.py:323
  /usr/local/lib/python3.11/site-packages/pydantic/_internal/_config.py:323: PydanticDeprecatedSince20: Support for clas
s-based `config` is deprecated, use ConfigDict instead. Deprecated in Pydantic V2.0 to be removed in V3.0. See Pydantic
V2 Migration Guide at https://errors.pydantic.dev/2.11/migration/
    warnings.warn(DEPRECATION_MESSAGE, DeprecationWarning)

../usr/local/lib/python3.11/site-packages/passlib/utils/__init__.py:854
  /usr/local/lib/python3.11/site-packages/passlib/utils/__init__.py:854: DeprecationWarning: 'crypt' is deprecated and s
lated for removal in Python 3.13
    from crypt import crypt as _crypt

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
========================================== 1 passed, 4 warnings in 1.00s ==========================================
```

Figura 8: Criação de Usuario e geração de email institucional
Fonte: Elaboração própria (2025)

Caso de teste 7 (CT-07) - Lógica de Severidade de Logs

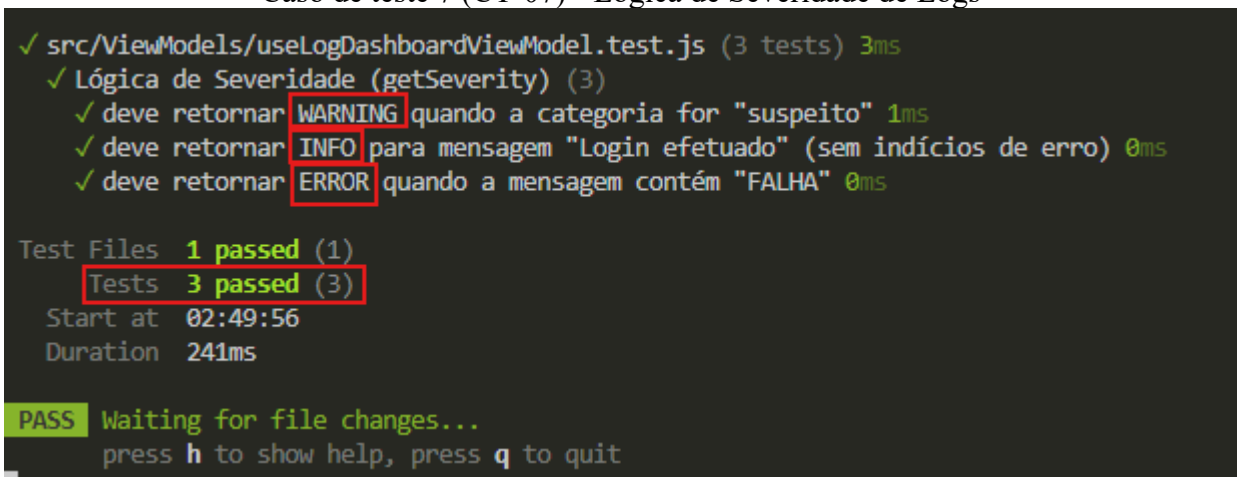

Figura 9: Prompt de retornos de severidade
Fonte: Elaboração própria (2025)

Caso de Testes (CT-08) - Integridade Referencial Usuário-Perfil

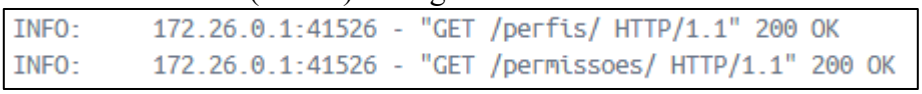

Figura 10 – Requisição GET para atribuição de permissão ao respectivo perfil
Fonte: Elaboração própria(2025)

Caso de Testes (CT-08) - Integridade Referencial Usuário-Perfil

Figura 11 – Evidência das respectivas permissões relacionadas ao perfil criado
Fonte: Elaboração própria (2025)

Caso de teste x (CT-0x) – Deletar usuários existentes

```
(.venv) PS D:\OneDrive\Documentos\GitHub\SolucoesComputacionais-ArgosForense\Analisador-Forense\argos> docker-compose ex
ec api_gestao python -m pytest tests/unit/test_user_delete.py -v
time="2025-11-27T02:23:45-03:00" level=warning msg="D:\\OneDrive\\Documentos\\GitHub\\SolucoesComputacionais-ArgosForens
e\\Analisador-Forense\\argos\\docker-compose.yml: `version` is obsolete"
============================================= test session starts =============================================
platform linux -- Python 3.11.14, pytest-7.4.3, pluggy-1.6.0 -- /usr/local/bin/python
cachedir: .pytest_cache
rootdir: /app
plugins: asyncio-0.23.5, anyio-4.11.0
asyncio: mode=Mode.STRICT
collected 1 item

tests/unit/test_user_delete.py::test_deletar_usuario_existente PASSED                                    [100%]

============================================== warnings summary ===============================================
../usr/local/lib/python3.11/site-packages/pydantic/_internal/_config.py:323
../usr/local/lib/python3.11/site-packages/pydantic/_internal/_config.py:323
../usr/local/lib/python3.11/site-packages/pydantic/_internal/_config.py:323
  /usr/local/lib/python3.11/site-packages/pydantic/_internal/_config.py:323: PydanticDeprecatedSince20: Support for clas
s-based `config` is deprecated, use ConfigDict instead. Deprecated in Pydantic V2.0 to be removed in V3.0. See Pydantic
V2 Migration Guide at https://errors.pydantic.dev/2.11/migration/
    warnings.warn(DEPRECATION_MESSAGE, DeprecationWarning)

../usr/local/lib/python3.11/site-packages/passlib/utils/__init__.py:854
  /usr/local/lib/python3.11/site-packages/passlib/utils/__init__.py:854: DeprecationWarning: 'crypt' is deprecated and s
lated for removal in Python 3.13
    from crypt import crypt as _crypt

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
========================================== 1 passed, 4 warnings in 0.82s ==========================================
```

Caso de teste 9 (CT-09) – Integração do Pipeline de Ingestão

```
StatusCode        : 200
StatusDescription : OK
Content           : []

RawContent        : HTTP/1.1 200 OK
                    Access-Control-Allow-Origin: *
                    Connection: close
                    Content-Length: 3
                    Content-Type: application/json
                    Date: Thu, 27 Nov 2025 06:27:21 GMT
                    Server: Werkzeug/3.1.3 Python/3.12.3

                    []

Forms             : {}
Headers           : {[Access-Control-Allow-Origin, *], [Connection, close], [Content-Length, 3], [Content-Type,
                    application/json]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 3
```

Figura 12 – Erro de Integração conteúdo vázio '[ ]'
Fonte: Elaboração própria (2025)

Caso de teste (CT-10) - Ciclo de Vida de Gestão de Usuários

```
INFO:     172.26.0.1:44600 - "POST /usuarios/69280417dc3ad33ec22be9d7/desativar HTTP/1.1" 200 OK
INFO:     172.26.0.1:44600 - "GET /usuarios/ HTTP/1.1" 200 OK
INFO:     172.26.0.1:42938 - "POST /usuarios/69280417dc3ad33ec22be9d7/ativar HTTP/1.1" 200 OK
INFO:     172.26.0.1:42938 - "GET /usuarios/ HTTP/1.1" 200 OK
```

Figura 13 – Rota integra
Fonte: Elaboração própria (2025)

Caso de teste (CT-11) - Recuperação Automática de Falha no Banco

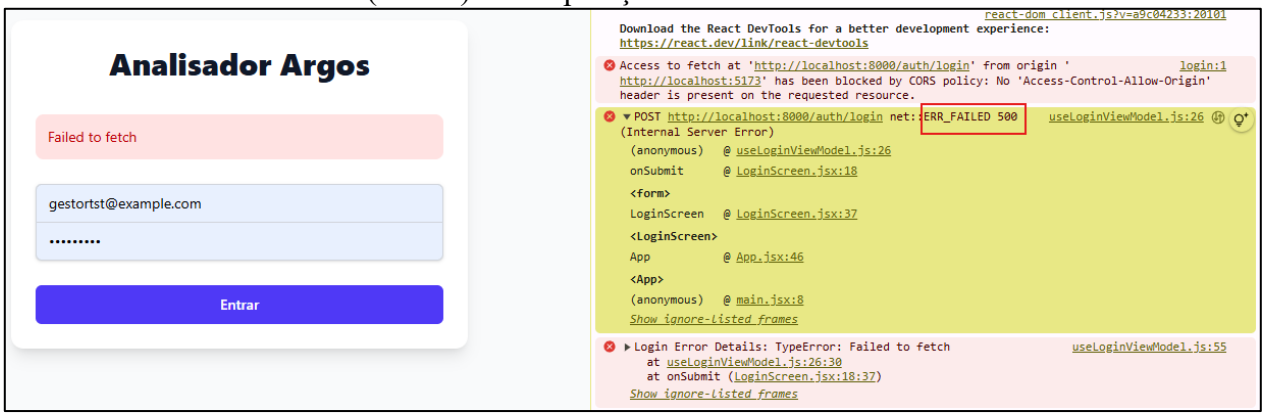

Figura 14 – Erro de Integração conteúdo vázio '[ ]'
Fonte: Elaboração própria (2025)

Caso de teste (CT-12) - Teste de Stress e Carga (Performance)

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
b0c3fad548bc	api_gestao	4.91%	79.43MiB / 7.724GiB	1.00%	1.95MB / 885kB	0B / 0B	11
b5ff061028df	api_logs	0.01%	35.16MiB / 7.724GiB	0.44%	7.5MB / 7.17MB	0B / 0B	1
e137bb9a3aa3	analisador	0.00%	21.84MiB / 7.724GiB	0.28%	15.6MB / 10.2MB	0B / 0B	1
e6c6ca5d5949	filebeat	0.04%	66.35MiB / 7.724GiB	0.84%	4.38MB / 15MB	0B / 0B	18
a14b44a5892a	elasticsearch	0.25%	1.193GiB / 7.724GiB	15.45%	26.4MB / 26.7MB	0B / 0B	131
90a57c59dba6	mongo	0.33%	111.2MiB / 7.724GiB	1.41%	1.59MB / 3.27MB	0B / 0B	47
2d66219d5507	gerador_logs	0.02%	8.246MiB / 7.724GiB	0.10%	746B / 0B	0B / 0B	3
CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
b0c3fad548bc	api_gestao	4.91%	79.43MiB / 7.724GiB	1.00%	1.95MB / 885kB	0B / 0B	11
b5ff061028df	api_logs	0.01%	35.16MiB / 7.724GiB	0.44%	7.5MB / 7.17MB	0B / 0B	1
e137bb9a3aa3	analisador	0.00%	21.84MiB / 7.724GiB	0.28%	15.6MB / 10.2MB	0B / 0B	1
e6c6ca5d5949	filebeat	0.04%	66.35MiB / 7.724GiB	0.84%	4.38MB / 15MB	0B / 0B	18
a14b44a5892a	elasticsearch	0.25%	1.193GiB / 7.724GiB	15.45%	26.4MB / 26.7MB	0B / 0B	131
90a57c59dba6	mongo	0.33%	111.2MiB / 7.724GiB	1.41%	1.59MB / 3.27MB	0B / 0B	47
2d66219d5507	gerador_logs	0.02%	8.246MiB / 7.724GiB	0.10%	746B / 0B	0B / 0B	3

Figura 15 - Performance da máquina
Fonte: Elaboração própria (2025)

Caso de teste (CT-12) - Teste de Stress e Carga (Performance)

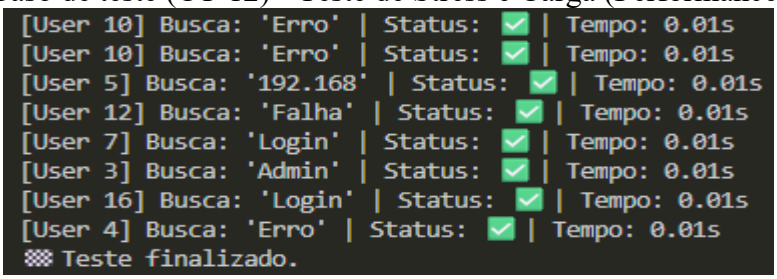

Figura 16 – 5 instâncias de gerador = x5 velocidade de gravação de logs
Fonte: Elaboração (2025)

Caso de teste 13 (CT-13) - Segurança - Tentativa de Injeção NoSQL

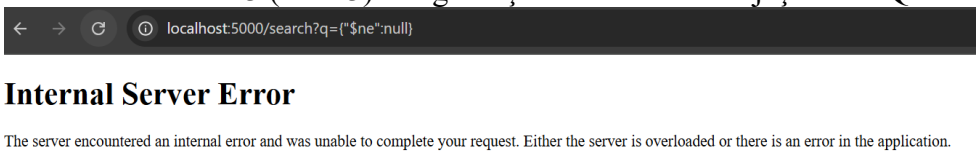

Figura 17 – Print retorno de pagina não encontrada
Fonte: Elaboração (2025)

Caso de teste 14 (CT-14) - Avaliação Heurística da Interface: Visibilidade

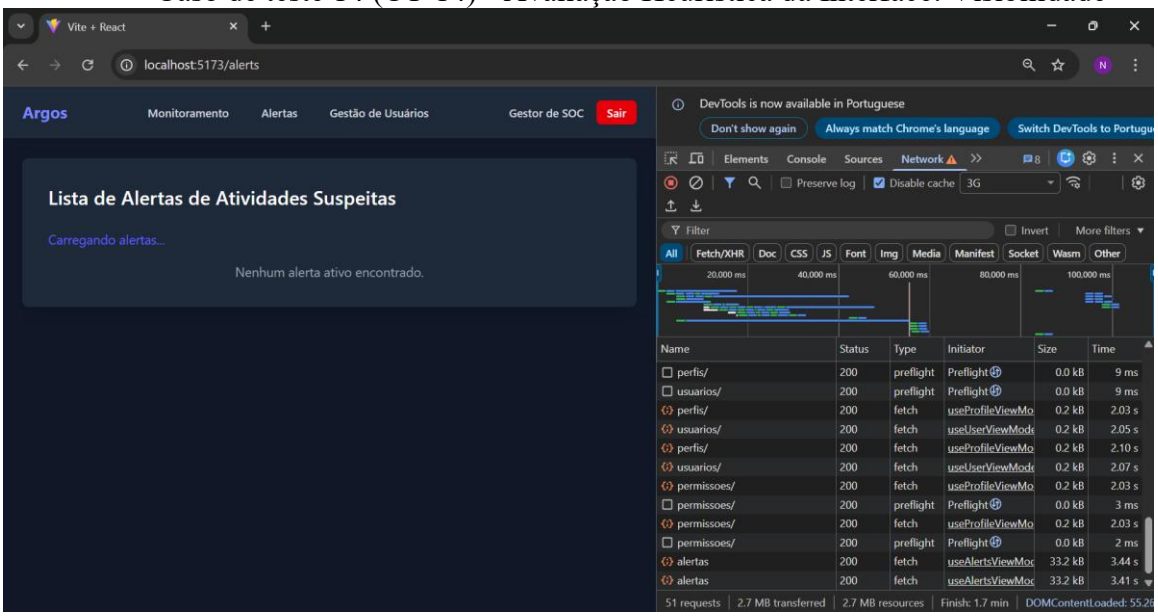

Figura 18 – Visibilidade: Forçando um carregamento lento
Fonte: Elaboração (2025)

Caso de teste 14 (CT-14) - Avaliação Heurística da Interface: Prevenção

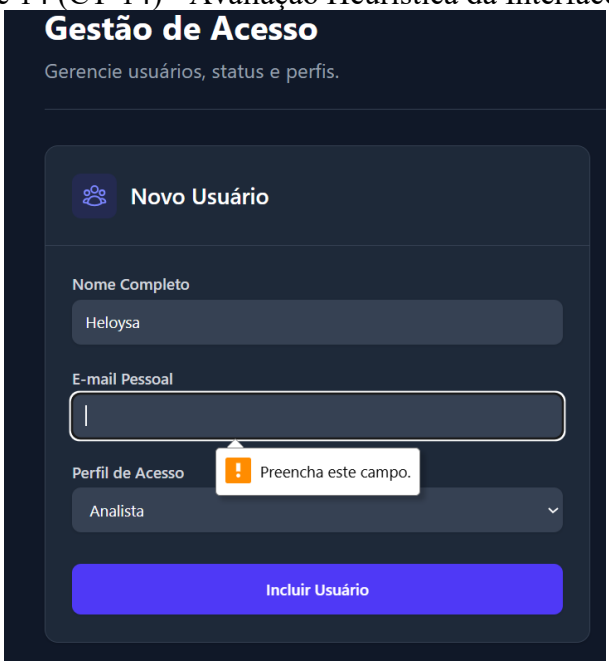

Figura 19 – Prevenção: Indicação de campo vazio
Fonte: Elaborada pelo autor (2025)

Caso de teste 14 (CT-14) - Avaliação Heurística da Interface: Terminologia

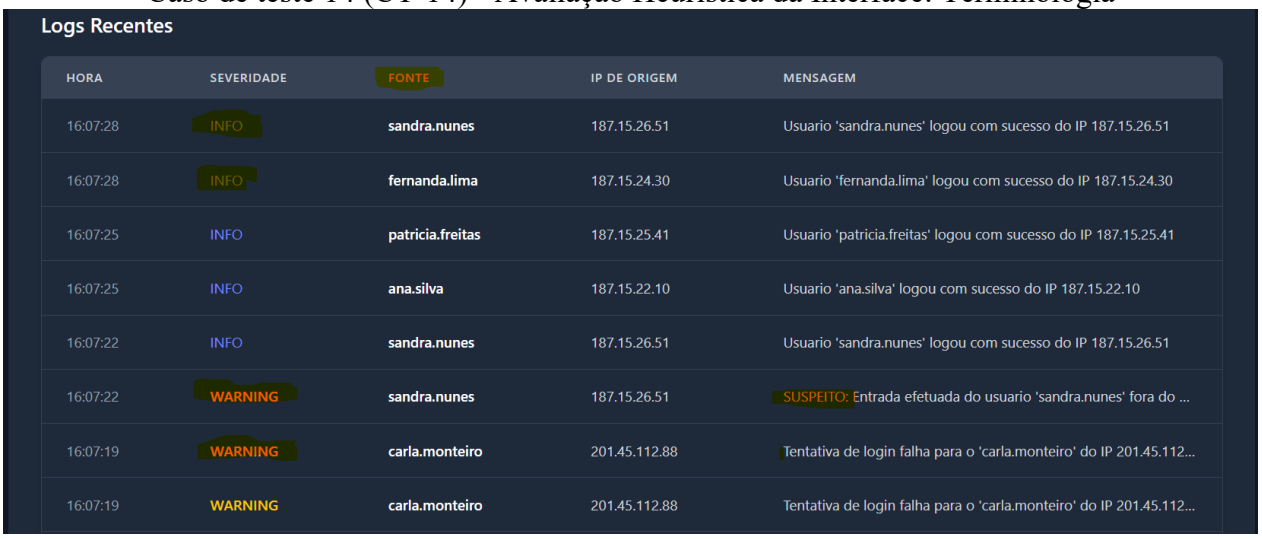

Figura 20 – Terminologia no painel de monitoramento
Fonte: Elaborada pelo autor (2025)

Caso de teste 15 (CT-15) - Responsividade e Compatibilidade Visual

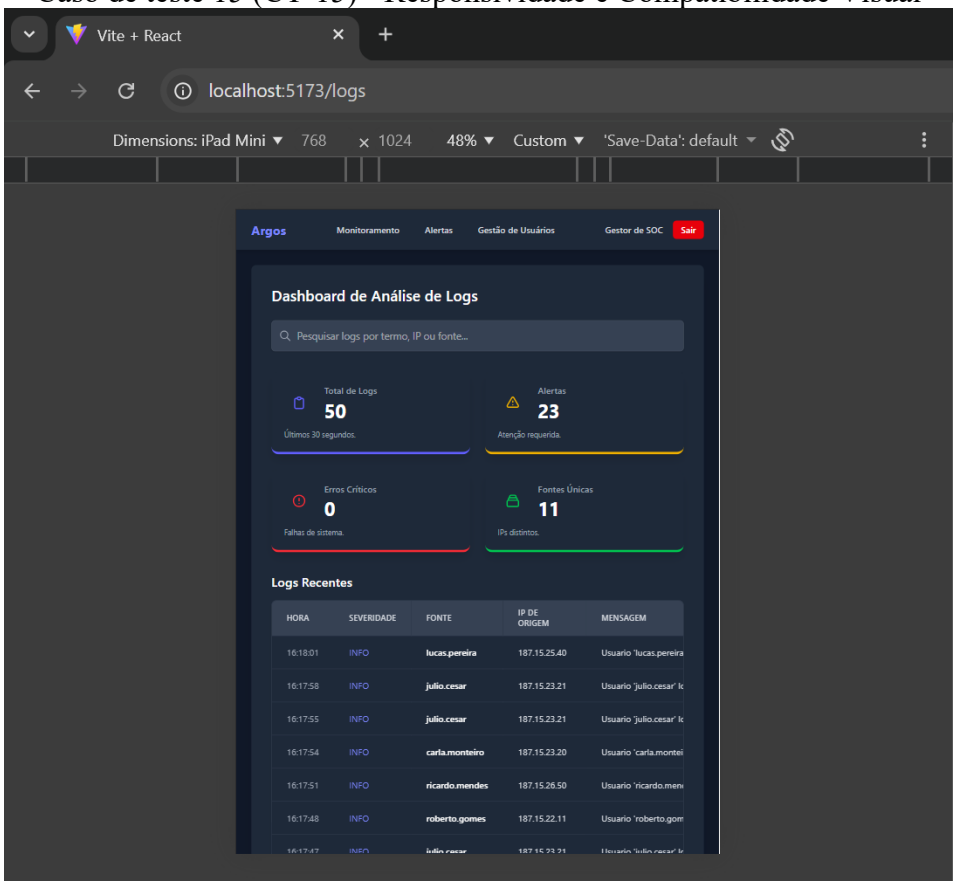

Figura 21 – Ecrã de tablet (largura 768px)
Fonte: Elaborada pelo autor (2025)

Caso de teste 16 (CT-16) - Sessão de Teste Exploratório

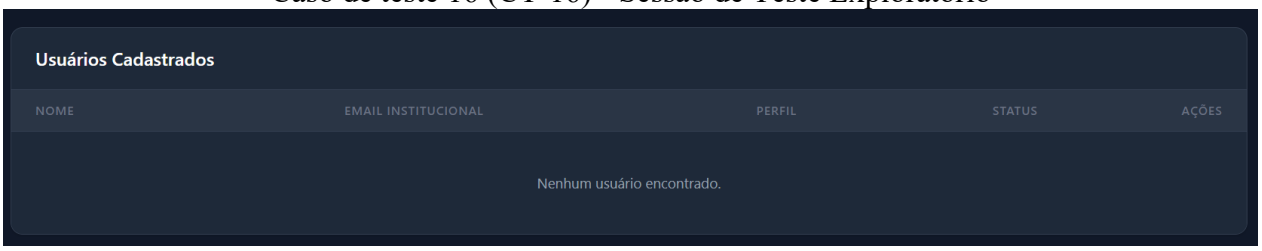

Figura 22 – Mensagem de nenhum usuário encontrado
Fonte: Elaborada pelo autor (2025)