MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAIANO - CAMPUS CATU
CURSO TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ARTHUR SALDANHA FÉLIX ULISSES

Desenvolvimento de uma Interface Web para o Monitoramento de Variáveis Ambientais e Segurança em Protótipo de Colmeia Inteligente

CATU-BAHIA

2025

ARTHUR SALDANHA FÉLIX ULISSES

Desenvolvimento de uma Interface Web para o Monitoramento de Variáveis Ambientais e Segurança em Protótipo de Colmeia Inteligente

Trabalho de Conclusão de Curso apresentado ao curso de Análise e Desenvolvimento de Sistemas como requisito parcial para à obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas pelo Instituto Federal de Educação, Ciência e Tecnologia Baiano campus Catu.

Orientador(a): Prof. Dr. Gilvan Martins Durães

CATU-BAHIA

2025

Ficha catalográfica elaborada pelo Bibliotecário Ricardo Santos do Carmo Reis - CRB – 5ª / 1649

Ulisses, Arthur Saldanha Félix

U39d Desenvolvimento de uma interface Web para o monitoramento de variáveis ambientais e segurança em protótipo de colmeia inteligente/ Arthur Saldanha Félix Ulisses.- Catu, BA, 2025.

65p.; il.: color.

Inclui bibliografia.

Trabalho de Conclusão de Curso (Graduação em Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia Baiano – Campus Catu.

Orientador: Prof. Dr. Gilvan Martins Durães.

1. Apicultura de precisão. 2. Internet das coisas. 3. Monitoramento de colmeias. I. Instituto Federal de Educação, Ciência e Tecnologia Baiano. II. Durães, Gilvan Martins (Orient.). III. Título.

CDU: 004.5:638.1

Agradecimentos

A Deus, pela vida, pela força e por iluminar meus passos ao longo desta caminhada.

Aos meus pais, em especial a minha mãe, pelo apoio incondicional aos estudos e por acreditar no meu potencial em cada etapa.

Ao meu orientador, cuja sabedoria e conhecimento foram essenciais em diversos momentos do curso, oferecendo direção, rigor e confiança para a concretização deste trabalho.

Aos colegas de turma, pela convivência e colaboração que tornaram o percurso mais leve e significativo.

Aos(às) servidores(as) desta instituição, pelo profissionalismo e dedicação que sustentam o nosso cotidiano acadêmico.

Por fim, a mim mesmo, pela persistência e por não desistir diante dos desafios.

“O futuro pertence àqueles que acreditam em seu próprio esforço.”

(Autor desconhecido)

Resumo

O manejo de apiários no Brasil lida com questões complexas, como a necessidade de intervenções menos invasivas e a crescente preocupação com a segurança patrimonial contra furtos. Neste contexto, a Apicultura de Precisão, utilizando a Internet das Coisas (IoT), apresenta-se como uma abordagem tecnológica viável. Este trabalho, inserido no "Projeto Melissa" do IF Baiano, teve como objetivo geral desenvolver uma interface *web* funcional para um protótipo de colmeia inteligente, capaz de receber, armazenar e exibir dados de sensores ambientais (temperatura, umidade, peso, luminosidade, uv) e de segurança (alertas de abertura) em tempo real. A metodologia adotada foi a *Design Science Research* (DSR), focada na construção de um artefato de *software*. O sistema foi desenvolvido com tecnologias de código aberto (*PHP* 8 e *MySQL*) e é composto por uma API de ingestão, que recebe dados de um módulo *ESP8266*, e uma interface *web* responsiva com *dashboards* e gráficos. Os resultados demonstraram, através de testes de API (caixa-preta) com a ferramenta Postman, o funcionamento robusto do *backend*, validando o recebimento de dados com baixa latência, a geração imediata de alertas de segurança e o registro de falhas de autenticação. Testes funcionais comprovaram a aderência da interface aos requisitos de controle de acesso baseado em papéis, responsividade e exportação de dados. O trabalho conclui entregando um artefato de *software* validado, de baixo custo e escalável, que atinge todos os objetivos propostos e contribui para a pesquisa em apicultura de precisão.

Palavras-chave: Apicultura de Precisão; Internet das Coisas; Monitoramento de Colmeias

Abstract

Apiary management in Brazil deals with complex issues, such as the need for less invasive interventions and growing concerns about property security against theft. In this context, Precision Beekeeping, using the Internet of Things (IoT), presents itself as a viable technological approach. This work, part of IF Baiano's "Melissa Project", had as its general objective to develop a functional *web* interface for a smart beehive prototype, capable of receiving, storing, and displaying real-time data from environmental sensors (temperature, humidity, weight, luminosity, uv) and security sensors (opening alerts). The methodology adopted was Design Science Research (DSR), focusing on the construction of a *software* artifact. The system was developed with open-source technologies (*PHP 8* and *MySQL*) and is composed of an ingestion API, which receives data from an *ESP8266* module, and a responsive *web* interface with *dashboards* and graphs. The results demonstrated, through API (black-box) tests with the Postman tool, the robust operation of the backend, validating data reception with low latency, the immediate generation of security alerts, and the registration of authentication failures. Functional tests confirmed the interface's adherence to requirements for role-based access control, responsiveness, and data export. The work concludes by delivering a validated, low-cost, and scalable *software* artifact that achieves all proposed objectives and contributes to research in precision beekeeping.

Keywords: Precision Beekeeping; Internet of Things; Beehive Monitoring.

Lista de Figuras

Sumário

1. INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

A biodiversidade e a produção agrícola dependem diretamente das abelhas. Elas são responsáveis por algo em torno de 40% da polinização ecossistêmica (RICKETTS et al., 2008). No Brasil, a apicultura transcende a relevância ambiental, firmando-se também como um pilar econômico, cenário que foi impulsionado pela chegada das abelhas africanizadas, um híbrido adaptado ao clima tropical que hoje domina a produção nacional (SANTOS, 2015).

1.2 PROBLEMA

A sobrevivência dessas abelhas, no entanto, está sob crescente ameaça. Questões como desmatamento, uso indiscriminado de agrotóxicos e manejo incorreto vêm sendo associados a um declínio das populações de abelhas, já relatados na literatura (Viana e SILVA, 2010; NOCELLI et al., 2012). Além disso, o próprio estresse das intervenções humanas frequentes prejudica o comportamento e a produtividade das colônias. Nesse cenário, a Apicultura de Precisão surge como uma alternativa viável. Utilizando a Internet das Coisas (IoT), o monitoramento remoto e contínuo passa a permitir uma gestão mais ágil, menos invasiva e com respostas rápidas a riscos (COTA et al., 2023). Estudos recentes confirmam: o uso de sensores para medir variações de temperatura, umidade ou peso é eficaz para identificar anomalias em tempo real, fundamentando uma tomada de decisão mais precisa (ZACEPINS et al., 2015; KULYUKIN et al., 2018).

Paralelamente aos desafios biológicos e de manejo, um problema de ordem prática tem se intensificado: a segurança dos apiários. O aumento de relatos de furtos e vandalismo em colmeias (FONTENELE, 2022) impõe a necessidade de integrar, de forma acessível e eficiente, mecanismos de monitoramento ambiental e de segurança patrimonial.

1.3 JUSTIFICATIVA

Este trabalho se insere no contexto descrito ao propor o desenvolvimento de uma interface *web* dedicada ao monitoramento e segurança de um protótipo de colmeia inteligente. A solução é parte integrante do projeto Melissa, desenvolvido no Instituto Federal Baiano Campus Catu. O autor deste TCC atuou como bolsista no referido projeto, participando ativamente do desenvolvimento do protótipo físico (o módulo de colmeia inteligente) que serve como base para este trabalho. O protótipo físico existente já é capaz de aferir variáveis críticas (temperatura, umidade, luminosidade, radiação UV e peso) e emitir alertas de abertura não autorizada. Esta pesquisa foca, portanto, em transformar essas medições brutas em um sistema *web* estruturado. Esse sistema é capaz de armazenar, exibir e analisar as informações de modo organizado, seguro e acessível. O objetivo dessa interface é oferecer uma visualização de dados clara e integrada, útil tanto para pesquisadores quanto para apicultores. Ela servirá, ainda, como alicerce para futuras aplicações em campo, ajudando a fomentar o uso de tecnologias acessíveis na apicultura de precisão.

A fundamentação desta proposta consiste integrar, de modo prático, os dados do protótipo físico a um ambiente digital que permita acompanhamento remoto e registro histórico. A construção da interface *web* viabiliza a análise das variáveis e a gestão dos alertas de segurança sem exigir intervenção física no apiário. Optou-se por tecnologias de ampla adoção e código aberto (*PHP, MySQL* e o módulo *ESP8266*) para reforçar o caráter acessível e educacional do projeto, o que facilita sua replicação ou aprimoramento em outros contextos.

1.4 OBJETIVO GERAL

O objetivo geral deste trabalho é criar uma interface *web* para um protótipo de colmeia inteligente que lide com os dados dos sensores em tempo real, desde o recebimento até o armazenamento e a exibição.

1.5 OBJETIVOS ESPECÍFICOS

a) levantar os requisitos funcionais e não funcionais da interface *web* proposta;

b) projetar e implementar o banco de dados que armazenará as informações ambientais e de segurança;

c) desenvolver a interface *web* responsiva para permitir a visualização dos registros.

d) integrar o protótipo físico com a aplicação *web*, garantindo que a comunicação entre o *hardware* e o *software* seja contínua.

O alcance desses objetivos visa consolidar um sistema funcional que une automação, análise de dados e segurança. Espera-se, com isso, contribuir para o avanço das pesquisas em apicultura inteligente, demonstrando o potencial de tecnologias IoT para promover simultaneamente o bem-estar das abelhas e a sustentabilidade no manejo dos apiários.

1.6 TRABALHOS CORRELATOS

A investigação da IoT aplicada à apicultura não é inédita, e diversos trabalhos recentes pavimentaram esse caminho. Estudos como os de SILVA (2017) focaram no monitoramento não invasivo; ZACEPINS et al. (2020) desenvolveram sistemas para aferir o estado da colônia (temperatura, peso, som); e propostas mais recentes, como as de CARDOSO et al. (2023) e MARQUES et al. (2024), validaram o uso de protótipos de baixo custo e arquiteturas IoT para coleta de dados ambientais. O presente trabalho se apoia nessas fundações, mas se diferencia ao propor uma abordagem que integra, em uma única plataforma, o monitoramento ambiental e a segurança física da colmeia. Ao unificar a coleta automatizada de dados, os alertas de segurança em tempo real e uma interface *web* dedicada, o projeto Melissa busca ampliar as possibilidades de pesquisa e ensino, demonstrando o potencial da tecnologia aplicada à sustentabilidade e à proteção de ecossistemas essenciais.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 FUNDAMENTOS DA APICULTURA E MELIPONICULTURA

A apicultura é uma prática humana antiga, notável por sua sustentabilidade. Ela não se resume à produção de alimentos ou derivados; tendo relevância também na manutenção da biodiversidade. No Brasil, a atividade ganhou contornos próprios, principalmente após a introdução das abelhas africanizadas (*Apis mellifera*). Esses insetos, um resultado do cruzamento de subespécies europeias e africanas, mostraram-se incrivelmente resistentes e adaptados ao nosso clima tropical (SILVA et al., 2021). Hoje, por causa da sua alta produtividade e rusticidade, essas "abelhas africanizadas" simplesmente dominam a produção nacional, viabilizando a apicultura em quase todos os biomas.

A Embrapa Meio-Norte (2021) reforça um ponto crucial: a apicultura vai além do seu valor econômico óbvio, sendo uma ferramenta de conservação ambiental. Afinal, são as abelhas que fazem a maior parte do trabalho de polinização, tanto em culturas agrícolas quanto na flora silvestre. Isso significa que elas ajudam a regenerar ecossistemas e dão estabilidade à nossa produção agrícola. Existe, portanto, uma interdependência clara entre a atividade apícola e o meio ambiente. Ela se torna essencial para o equilíbrio ecológico e a sustentabilidade no campo, sobretudo em locais com muita flora.

Nada disso funciona sem o cuidado correto. O sucesso da produção depende totalmente de um manejo adequado das colmeias. A obra Apicultura: criação de abelhas e produção de mel (GERONET SERVICES, 2004) é enfática sobre a importância de boas práticas: higienizar as caixas, manter um controle sanitário das colônias e, acima de tudo, respeitar o ciclo natural dos insetos. São essas ações que garantem não só a qualidade do mel, mas o bem-estar da colônia, o que por sua vez reduz perdas e evita o abandono das caixas.

Além da apicultura tradicional, outra atividade tem ganhado espaço: a meliponicultura. Trata-se da criação de abelhas nativas sem ferrão, como as

dos gêneros Melipona e Trigona. Elas produzem um volume de mel bem menor, se comparadas às africanizadas, mas sua relevância ecológica é única. Essas abelhas nativas conseguem polinizar espécies de plantas que as abelhas exóticas simplesmente não visitam. Como bônus, seu mel tem um valor comercial altíssimo, muito reconhecido por suas qualidades terapêuticas e sensoriais.

Por esses motivos, órgãos como a Embrapa têm incentivado a meliponicultura. Ela é vista como uma excelente alternativa de renda, especialmente para a agricultura familiar, ao mesmo tempo em que ajuda a conservar os ecossistemas locais. Fica claro que ambas as práticas, apicultura e meliponicultura, são fundamentais para um modelo de desenvolvimento rural sustentável, que une produção econômica e preservação.

2.2 APICULTURA DE PRECISÃO

Entender a base biológica e produtiva é o primeiro passo para modernizar o setor. É exatamente nesse ponto que a tecnologia pode contribuir. A integração de sensores, plataformas digitais e sistemas *web*, como a que este trabalho propõe, aparece como uma estratégia inovadora. O objetivo é usar essas ferramentas para promover um manejo mais inteligente e sustentável das colmeias, melhorando a eficiência da produção e a segurança geral dos apiários.

A apicultura de precisão é o que acontece quando o manejo convencional encontra a tecnologia. A ideia central é usar recursos tecnológicos para transformar a gestão das colmeias, buscando mais eficiência e sustentabilidade. Acima de tudo, o manejo passa a ser orientado por dados.

Com isso, o apicultor ganha uma nova capacidade: ele pode entender melhor o comportamento das abelhas. Anomalias são identificadas muito mais cedo. No fim, as decisões se tornam mais assertivas, pois são baseadas em informações quantitativas, não apenas na observação casual (ZACEPINS et al., 2015).

Nesse contexto, utilizam-se tecnologias emergentes, incluindo sensores eletrônicos, redes sem fio, sistemas embarcados e plataformas de visualização.

O uso desses dispositivos permite o acompanhamento detalhado constante das variáveis internas da colmeia. Tal monitoramento possibilita identificar anomalias precocemente, favorecendo tomadas de decisão rápidas que assegurem a sustentabilidade e a produtividade do apiário.

No Brasil, nosso desafio é adaptar essas tecnologias. Elas precisam funcionar em nossas condições climáticas e, o mais importante, caber na realidade socioeconômica. A busca é por soluções de baixo custo e manutenção simples. É exatamente nesse ponto que o uso de microcontroladores e módulos acessíveis, como o *ESP8266*, se mostra um caminho viável. COTA et al. (2023) destacam que essa abordagem funciona bem para criar protótipos em ambientes acadêmicos e comunitários. Isso acaba fomentando tanto a pesquisa aplicada quanto a formação de novos profissionais para a inovação no campo. O resultado prático é a possibilidade de monitorar muitas colmeias de longe, o que otimiza recursos e economiza deslocamentos.

A importância de monitorar as variáveis microclimáticas de forma contínua é um ponto-chave. Isso permite ao apicultor ajustar seu manejo às condições do ambiente, o que reduz o estresse das colônias e previne perdas significativas, como reforça MACHADO (2024). O autor também salienta os ganhos em eficiência operacional e práticas mais sustentáveis. Essa ideia de integração é levada adiante por MELO (2025), que apresentou um protótipo de colmeia inteligente com sensores de temperatura, umidade e peso, mas que também incluía alarmes automatizados para a segurança do apiário. Para ele, essa combinação de *hardware* e *software* é um passo importante para disseminar tecnologias que sejam acessíveis e replicáveis, especialmente em ambientes de pesquisa e educação.

Assim, a apicultura de precisão é uma área de convergência entre tecnologia e biologia. O conhecimento empírico do apicultor não é substituído; ele é enriquecido por informações vindas de dispositivos inteligentes. Essa união entre a coleta automatizada de dados e a análise digital acaba por promover um novo paradigma de manejo: observação contínua, no diagnóstico antecipado e em intervenções baseadas em evidências. É exatamente nesse

cenário que sistemas como o Melissa surgem, representando um avanço real na democratização da inovação tecnológica para a apicultura sustentável no Brasil.

2.3 INTERNET DAS COISAS (IOT) E A APRENDIZAGEM BASEADA EM PROJETOS

A Internet das Coisas (IoT) descreve como dispositivos físicos interagem. Eles são equipados com sensores, atuadores e conectividade.

Desse modo, quando usam a internet, conversam entre si e com plataformas digitais. Essa integração contínua entre o físico e o virtual é, para muitos, uma das evoluções mais significativas do nosso tempo (ATZORI, IERA e MORABITO, 2010)

Sistemas de sensoriamento distribuídos, que fornecem informações precisas e contínuas sobre o ambiente, são viabilizados justamente pela IoT (GUBBI et al., 2013). Quando redes de sensores se ligam a plataformas *web*, é possível criar soluções que cobrem o ciclo todo. Captura, processamento e visualização. O resultado final favorece tecnologias sustentáveis que ajudam na tomada de decisão.

Essa estratégia de monitoramento ambiental via IoT tem se provado eficiente. Ela serve para acompanhar variáveis climáticas e ecológicas de longe. Conectar sensores a uma infraestrutura sem fio é o que permite criar sistemas escaláveis. E esses sistemas geram informações em tempo real para muitos contextos. O manejo de colmeias, o controle agrícola e a gestão de recursos naturais são apenas alguns exemplos (ZANELLA et al., 2014).

Mas a IoT não é só uma ferramenta técnica; ela tem um enorme potencial para a aprendizagem ativa. Isso é muito relevante no meio educacional e científico. Um bom exemplo vem de MACIEL et al. (2017), que usaram a Aprendizagem Baseada em Projetos (PBL) para desenvolver um sistema de monitoramento ambiental. Os próprios estudantes participaram de todo o processo. Concepção, prototipagem e implementação da solução de coleta e visualização. A abordagem foi um sucesso. Além de desenvolver

competências técnicas, ampliou a compreensão prática dos alunos sobre eletrônica, programação e conectividade.

Essa experiência (MACIEL et al., 2017) demonstra algo importante: projetos de IoT podem ser, ao mesmo tempo, instrumentos de ensino e de pesquisa, unindo *hardware*, *software* e metodologias inovadoras. Essa perspectiva se alinha perfeitamente ao desenvolvimento do sistema Melissa. Nosso projeto segue esta linha, usando sensores conectados, Wi-Fi e armazenamento em servidor *web* para o acompanhamento remoto das variáveis e alertas de segurança. Ou seja, além da sua aplicação prática, o sistema carrega um forte componente didático e científico. Ele se alinha perfeitamente à formação interdisciplinar que o PBL promove.

No fim das contas, a Internet das Coisas aplicada ao monitoramento ambiental é uma tecnologia essencial. Ela serve tanto à pesquisa científica quanto à educação tecnológica. O uso dela em sistemas como o Melissa apenas evidencia esse potencial de integrar sustentabilidade, inovação e formação acadêmica. É um caminho para promover o uso responsável de tecnologias digitais no estudo e na preservação dos ecossistemas.

2.4 DESENVOLVIMENTO DE SISTEMAS WEB E BANCO DE DADOS

Hoje, o desenvolvimento de sistemas *web* é uma das principais áreas da engenharia de *software*. O foco é criar aplicações que rodam em navegadores e se conectam a servidores remotos. É uma abordagem que mistura princípios clássicos de engenharia com práticas de interoperabilidade, escalabilidade e, claro, experiência do usuário (PRESSMAN e MAXIM, 2021). Para sistemas que exigem acesso distribuído e processamento dinâmico, como os de IoT, ela se tornou fundamental.

Aplicações *web*, em sua maioria, usam a arquitetura cliente-servidor. A lógica é uma comunicação clara: o cliente, que costuma ser o navegador, faz uma requisição. O servidor, por sua vez, processa essa requisição e retorna um conteúdo ou dados. A grande vantagem dessa estrutura é a modularidade. Ela permite que as responsabilidades do sistema sejam divididas em camadas, como a apresentação (*frontend*), o negócio (backend) e os dados. Essa

separação, como aponta SOMMERVILLE (2019), é o que facilita a manutenção, o reuso de código e a segurança geral da informação.

Para o sistema Melissa, a camada de apresentação (*frontend*) foi construída com tecnologias padrão: HTML, CSS e JavaScript. Foi utilizado o framework leve PicoCSS e a biblioteca Chart.js, que é usada para exibir os dados visuais em tempo real. Essa escolha foi técnica. Ela atende aos requisitos de responsividade e acessibilidade do projeto. Isso permite que pesquisadores e usuários acessem o painel de qualquer dispositivo, seja computador, tablet ou smartphone.

A camada de negócio (*backend*) foi implementada em *PHP 8*. Essa camada atua como a ponte entre o banco de dados e a interface. Usamos o *PHP 8*, uma linguagem amplamente adotada em sistemas dinâmicos, por sua flexibilidade e simplicidade. Para uma aplicação educacional e de pesquisa como o Melissa, que exige baixo custo e alta portabilidade, foi uma escolha adequada (ULLMAN, 2017).

Para a integração com o banco de dados, fizemos a escolha técnica de usar a extensão *PDO* (*PHP Data Objects*). O *PDO* não é apenas uma forma de conectar ao *MySQL*; ele atua como uma camada de abstração que fornece uma interface única para diversos bancos. A principal vantagem dessa escolha é a segurança: o *PDO* facilita e incentiva o uso de prepared statements (consultas parametrizadas), que é a defesa mais robusta e recomendada contra ataques de *SQL Injection*. Isso garante que os dados vindos da interface ou da *API* não possam comprometer o banco de dados.

O núcleo de todo o sistema é o banco de dados. É ele quem guarda e recupera as informações. Nossa escolha foi o *MySQL*. Ele segue o modelo relacional clássico, proposto por CODD (1970). Nele, os dados são organizados em tabelas ligadas por chaves. Esse modelo garante consistência, integridade e um desempenho eficiente em consultas *SQL*. Não é à toa que os bancos relacionais continuam sendo amplamente usados em sistemas científicos e corporativos: sua estrutura lógica e robustez para gerenciar grandes volumes de dados são comprovadas (DATE, 2019).

No caso específico do Melissa, o banco de dados foi projetado para armazenar as leituras dos sensores (temperatura, umidade, luminosidade, UV e peso), além dos registros dos dispositivos e dos alertas de segurança. O fluxo funciona assim: o módulo *ESP8266* envia cada leitura ao servidor via uma requisição HTTP segura. Essa requisição contém um token de autenticação (uma *API key*) que identifica o dispositivo. O *backend* então processa a requisição, valida a autenticidade dos dados e, por fim, insere tudo nas tabelas corretas. Esse fluxo garante a rastreabilidade e a confiabilidade das informações, o que permite tanto a análise em tempo real quanto a geração de relatórios históricos.

Essa arquitetura *web* também foi projetada para permitir aprimoramentos futuros. Ela facilita a integração de novas funcionalidades, como o envio de notificações automáticas, um controle de permissões mais refinado ou a visualização de múltiplas colmeias. Essa estrutura modular reforça o caráter escalável e educativo do projeto, que pode ser expandido para outros contextos. O desenvolvimento do sistema Melissa, portanto, exemplifica a aplicação de princípios consolidados da engenharia de *software* e da modelagem de dados em uma solução prática, voltada à pesquisa, à sustentabilidade e à inovação tecnológica.

3. METODOLOGIA

Para o desenvolvimento do *software*, foi utilizada a abordagem Design Science Research (DSR), metodologia muito usada na computação e na engenharia de *software*, uma vez que ela foca na criação e validação de artefatos tecnológicos para resolver problemas reais (HEVNER et al., 2004; PEFFERS et al., 2007). Ela permite que a construção da interface *web* seja um processo científico e iterativo.

Os dados em tempo real utilizados neste trabalho foram fornecidos pelo protótipo de colmeia inteligente do projeto Melissa. A partir desses dados, foi desenvolvida e testada a interface *web* proposta. Dessa forma, as etapas metodológicas foram, assim, estruturadas: levantamento de requisitos, modelagem e desenvolvimento, e validação do *software*.

3.1 TECNOLOGIAS UTILIZADAS E JUSTIFICATIVA

A seleção do conjunto de tecnologias para o Sistema Melissa foi guiada diretamente pelos requisitos funcionais e não funcionais definidos no Apêndice A , com foco em soluções de código aberto, baixo custo, segurança e facilidade de implantação.

Para o desenvolvimento do *backend*, optou-se pela linguagem *PHP* 8. Esta escolha se justifica por sua ampla adoção no mercado, vasta documentação e, principalmente, pela facilidade de implementação em servidores de hospedagem compartilhada, como a Hostinger, o que reduz o custo de manutenção do projeto. A versão 8 foi especificamente escolhida por suas melhorias significativas de performance e recursos de tipagem mais fortes. Conforme detalhado na Fundamentação Teórica e implementado no *script ingest.php*, a integração com o banco de dados foi feita exclusivamente através da extensão *PDO* (*PHP Data Objects*). Esta foi uma decisão arquitetural focada em segurança, sendo a principal ferramenta do *PHP* para prevenir ataques de *SQL Injection* através de *prepared statements*, atendendo diretamente ao requisito RNF03.

Como sistema gerenciador de banco de dados, o *MySQL* foi selecionado, atendendo diretamente ao requisito RNF04 ("Banco relacional (*MySQL*)"). O *MySQL* é uma solução open-source, robusta e amplamente testada, capaz de gerenciar com eficiência os relacionamentos entre as tabelas de leituras, dispositivos e alertas, sendo totalmente compatível com o ambiente de hospedagem escolhido.

No que diz respeito à camada de apresentação (*frontend*), as tecnologias base (HTML5, CSS3 e JavaScript) foram complementadas por duas bibliotecas principais. Para a estilização, optou-se pelo PicoCSS. Diferente de frameworks mais pesados, o PicoCSS é uma estrutura CSS minimalista que foca em estilizar tags HTML semânticas, permitindo a criação de uma interface limpa e atendendo ao requisito RNF05 ("*Interface web* deve ser responsiva") com o mínimo de sobrecarga. Para a visualização de dados, a biblioteca Chart.js foi escolhida por sua simplicidade de integração com dados fornecidos pelo *backend* e sua capacidade de renderizar gráficos dinâmicos, cumprindo o requisito RF08 ("Exibir *dashboard* com... gráficos").

A arquitetura de implantação, descrita na Figura 4 (Seção 4.3.2), utiliza a plataforma de hospedagem Hostinger. Esta escolha se deu pelo seu baixo custo e por oferecer o ambiente *LAMP* (*Linux, Apache, MySQL, PHP*) completo e pré-configurado, necessário para a execução do *backend* (*api/ingest.php*) e do banco de dados , além de prover o certificado *SSL* para comunicação *HTTPS*, atendendo ao requisito RNF01.

Por fim, o *hardware* base (protótipo), desenvolvido em etapa anterior, consiste na plataforma Arduino Mega e no módulo *Wi-Fi ESP8266*. O Mega foi escolhido por sua grande quantidade de portas de I/O, necessárias para conectar os múltiplos sensores, enquanto o *ESP8266* é a solução padrão de mercado para prover conectividade IoT de baixo custo, sendo o ator responsável por enviar os dados ao servidor.

3.2 LEVANTAMENTO DE REQUISITOS

O levantamento de requisitos foi realizado com a elaboração de um Documento de Requisitos. Esse foi o passo para formalizar o escopo do

projeto, definir os atores e as regras do sistema. O escopo principal deste projeto é o desenvolvimento de um protótipo *web*. Esse protótipo coleta e armazena os dados dos sensores (por meio do *ESP8266*) e, em seguida,os disponibiliza em uma interface que funciona bem em qualquer tela.

Identificamos três atores principais para o sistema. O primeiro é o *Dispositivo (ESP8266)*, um ator não-humano que envia leituras e alertas. Os outros dois são humanos: o *Pesquisador*, que consome os dados (*dashboards*, gráficos), e o Administrador, um usuário com privilégios para gerenciar colmeias, dispositivos e os próprios usuários.

Com os atores definidos, classificamos os requisitos em funcionais, não funcionais e regras de negócio.

3.2.1 Requisitos Funcionais (RF)

Os requisitos funcionais definem o que o sistema de fato executa. Em vez de uma lista, agrupamos as funcionalidades por área. Para a gestão do sistema, definimos que o Administrador precisaria de módulos para Cadastrar colmeias (RF01) e Cadastrar dispositivos (RF02), sendo esta última responsável por gerar a *API key* única. Também incluímos um módulo completo de autenticação de usuários (RF07) para proteger as rotas internas.

Para a ingestão de dados, que é o núcleo do sistema, o *backend* precisaria Receber dados do ESP8266 (RF03) validando a *API key*, Persistir as leituras no banco de dados (RF04), e o mais importante, Registrar alertas de abertura não autorizada (RF05). Funções de apoio como Atualizar o status do dispositivo (RF06) e Manter logs de rejeição (RF10) também foram especificadas.

Por fim, para a apresentação ao usuário, o sistema deveria exibir um *dashboard* completo com gráficos e alertas (RF08) e ter uma função de exportação de dados em CSV (RF09).

3.2.2 Requisitos Não Funcionais (RNF)

Já os requisitos não funcionais focam nos atributos de qualidade do sistema, como ele opera. Demos atenção especial à segurança, definindo que toda comunicação usaria HTTPS (RNF01), as senhas seriam armazenadas

com hash seguro (RNF03), e o *endpoint* de ingestão teria rate limit (RNF06) e logs de falha (RNF07). Em desempenho, a *API* de ingestão precisaria ser rápida, com respostas abaixo de *500ms* (RNF02). Na tecnologia, especificamos o uso de um banco relacional *MySQL* (RNF04) e, na usabilidade, que a interface seria obrigatoriamente responsiva (RNF05).

3.2.3 Regras De Negócio (RB)

Por fim, as regras de negócio ditam a lógica específica do Melissa. Definimos que a *API key* é única e obrigatória (RB01) e que requisições sem ela devem ser rejeitadas (RB02). Além disso, apenas dispositivos ativos podem enviar dados (RB04). Duas regras de automação são fundamentais: o sistema deve gerar um alerta *CRITICAL* automaticamente se a colmeia for aberta (RB03), e o acesso a qualquer *dashboard* é restrito a usuários logados (RB05).

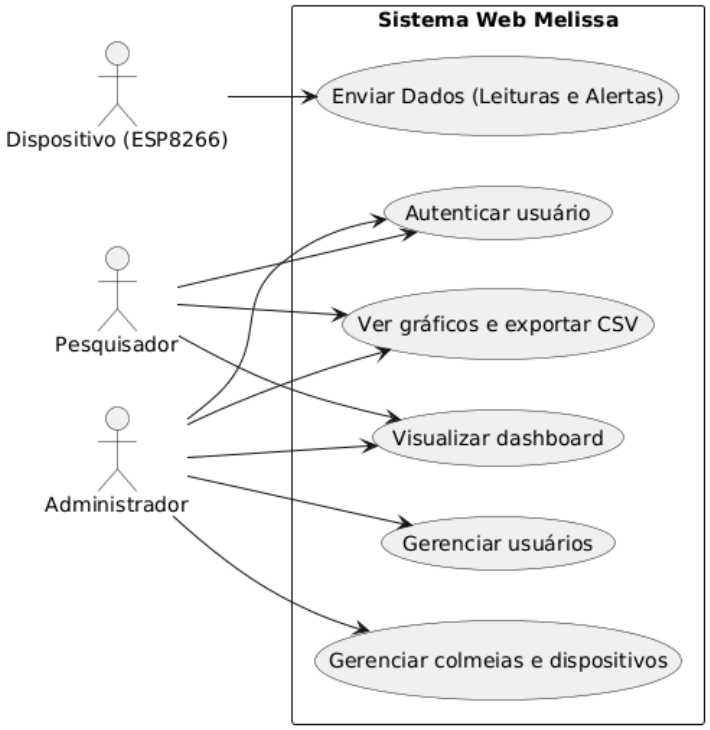
4. PROJETO E DESENVOLVIMENTO DO SISTEMA

Com base na fundamentação metodológica e nos requisitos levantados, este capítulo apresenta a etapa de construção do artefato tecnológico. São detalhadas as fases da engenharia de *software*, abrangendo a modelagem do sistema, a arquitetura da solução, o projeto de banco de dados e os detalhes da implementação.

4.1 MODELAGEM E CASOS DE USO

Com base nos requisitos funcionais e atores definidos na Seção 3.2, realizamos a modelagem visual das funcionalidades do sistema. Para isso, utilizou-se a notação UML (*Unified Modeling Language*) para criar o Diagrama de Caso de Uso (Figura 1), que ilustra as interações entre os atores e as principais funções do Sistema *Web* Melissa.

Figura 1 – Diagrama de Caso de Uso do Sistema Melissa



Fonte: O autor (2025)

O diagrama é estruturado em torno dos três atores *identificados* no documento de requisitos: o *Dispositivo (ESP8266)*, o *Pesquisador* e o *Administrador* .

O ator *Dispositivo* (*ESP8266*) é o agente não-humano que interage com um único caso de uso central: Enviar Dados (Leituras e Alertas). Este caso de uso encapsula toda a lógica de ingestão. Ele representa o envio do *payload* unificado para o *backend* (RF03) , que por sua vez é responsável por persistir as leituras no banco (RF04) e, com base no conteúdo do *payload*, gerar automaticamente o alerta de segurança (RF05, RB03).

O ator *Pesquisador* representa o usuário focado no consumo dos dados. O diagrama ilustra suas interações com os casos de uso *Autenticar usuário* (RF07) , *Visualizar dashboard* (RF08) e *Ver gráficos* e *exportar CSV* (RF09). A autenticação, conforme a regra de negócio RB05, é um pré-requisito para que o pesquisador possa acessar as telas de visualização e exportação.

Por fim, o ator *Administrador* é o usuário com privilégios de gestão. O diagrama demonstra que ele realiza todas as ações do *Pesquisador* (autenticar, visualizar e exportar). Adicionalmente, ele possui casos de uso exclusivos para a administração do sistema, que são: Gerenciar colmeias e dispositivos e Gerenciar Usuários. O primeiro caso de uso é a representação dos requisitos RF01 e RF02. O segundo caso, Gerenciar Usuários, atende à descrição do ator no documento de requisitos e ao requisito implícito de autenticação (RF07), permitindo que o administrador crie, edite ou remova as contas de outros usuários.

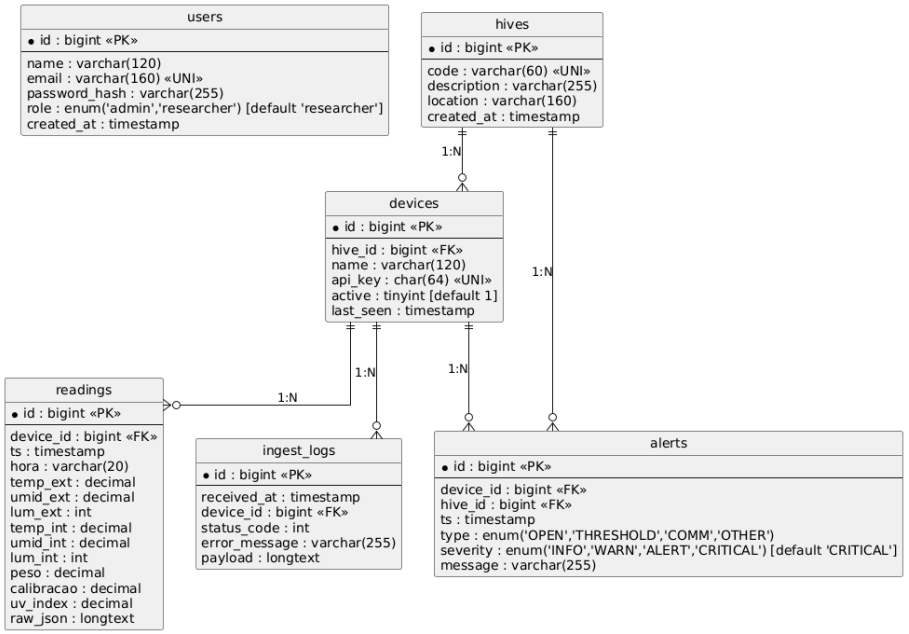
Essa modelagem foi crucial para validar o escopo e serviu como alicerce para o design da arquitetura e do banco de dados, que serão detalhados a seguir.

4.2 MODELAGEM DO BANCO DE DADOS

Seguindo o requisito não funcional RNF04, que previa o uso de um banco de dados relacional, optamos pela tecnologia *MySQL*. A etapa de modelagem foi essencial para estruturar como as informações seriam armazenadas, garantindo integridade e performance. O Modelo

Entidade-Relacionamento (DER) final do sistema é apresentado na Figura 2, a seguir.

Figura 2 – Modelo Entidade-Relacionamento (DER) do Sistema Melissa



Fonte: O autor (2025)

O modelo foi projetado para ser normalizado, garantindo a integridade referencial através do uso de *chaves primárias* (*«PK»*) e *estrangeiras* (*«FK»*). As tabelas centrais do sistema são descritas a seguir.

A tabela users é responsável por armazenar os dados de autenticação e permissão dos usuários, atendendo ao requisito RF07. Ela armazena o email como identificador único (*«UN»*), o *password_hash* (que armazena a senha com *hash* seguro, conforme RNF03) e o *role* (*enum* '*admin*' ou '*researcher*'), que define o nível de acesso do usuário.

As tabelas *hives* e *devices* gerenciam o cadastro do *hardware*. A tabela *hives* armazena os dados da colmeia física (RF01), como sua *location*. A tabela devices armazena os dados do módulo eletrônico (RF02), como a *api_key* («UN») usada para autenticar a *API* (RB01). O relacionamento de 1:N entre hives e devices (através da chave estrangeira *hive_id*) permite que uma colmeia possa, futuramente, ter múltiplos dispositivos de monitoramento.

A tabela readings é o núcleo de armazenamento de dados dos sensores, atendendo ao RF04. Ela possui um relacionamento 1:N com a tabela devices, indicando que um dispositivo gera múltiplas leituras ao longo do tempo. Esta tabela foi projetada para armazenar cada variável ambiental (como *temp_ext*, *umid_int*, *peso*, *uv_index*) em colunas dedicadas para facilitar consultas e a geração de gráficos, ao mesmo tempo em que armazena o *raw_json*, garantindo que nenhum dado bruto seja perdido.

A tabela *alerts* armazena os eventos de segurança (RF05). Ela possui chaves estrangeiras que a relacionam tanto a *devices* quanto a hives, permitindo consultas rápidas de alertas por colmeia ou por dispositivo. O campo type (enum '*OPEN*') e severity (enum '*CRITICAL*') implementam a regra de negócio RB03.

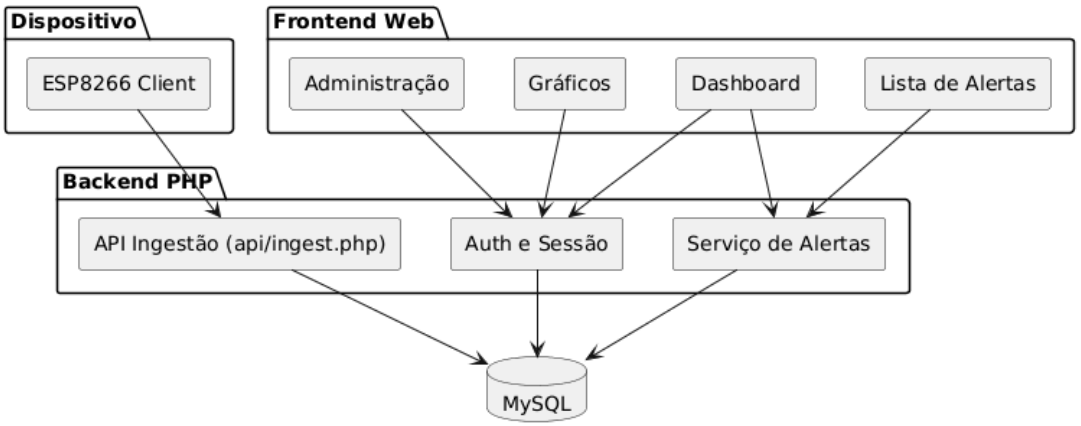
Finalmente, a tabela *ingest_logs* atende aos requisitos de monitoramento RNF07 e RF10. Ela registra todas as tentativas de ingestão de dados, sejam elas bem-sucedidas ou falhas. Colunas como *status_code* e *error_message* são essenciais para depuração e para identificar falhas de comunicação do *hardware* ou tentativas de acesso indevido.

4.3 ARQUITETURA DA SOLUÇÃO

4.3.1 Arquitetura De Componentes

Após a modelagem dos dados (Seção 4.2), o próximo passo foi definir a arquitetura lógica do *software*. Para isso, foi elaborado o Diagrama de Componentes (Figura 3), que ilustra os principais componentes de *software* do sistema Melissa e as dependências entre eles.

Figura 3 – Diagrama de Componentes do Sistema Melissa



Fonte: O autor (2025)

A arquitetura foi logicamente dividida em quatro pacotes ou camadas principais: *Dispositivo*, *Backend PHP*, *Frontend Web* e o banco de dados *MySQL*.

O pacote *Dispositivo* representa o *hardware* físico. Ele contém um único componente de *software*, o *ESP8266 Client*. Este componente é responsável por coletar os dados e reportar o *payload* unificado . Sua única dependência é o componente de ingestão no *backend*, com o qual se comunica via *HTTP POST*.

O pacote *Backend PHP* é o núcleo do sistema, onde residem as regras de negócio. Ele é composto por três componentes principais. O primeiro é a *API* Ingestão (*api/ingest.php*), uma interface que recebe os dados do *ESP8266 Client*. Ela é responsável por validar a *API* key (RB01, RB02) , persistir as leituras (RF04) e, crucialmente, gerar os registros de alerta (RF05) no *MySQL* com base no *payload* recebido (RB03). O segundo componente é o Auth e Sessão, o pilar central de segurança que implementa o requisito RF07. Ele é consumido por todos os componentes do *frontend* e se comunica com o *MySQL* para validar as credenciais do usuário. Por fim, o Serviço de Alertas é responsável por consultar e formatar os dados de alertas para o *frontend*, enquanto a *API* Ingestão cria os alertas, este serviço é consumido pelos componentes *Dashboard* e Lista de Alertas para ler e exibir os alertas (RF08).

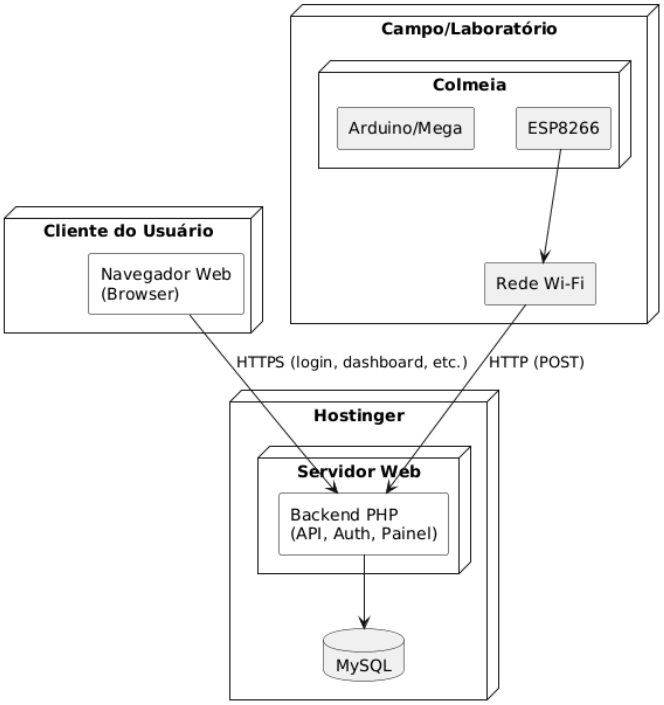
O pacote *Frontend Web* representa a camada de apresentação. Ele é composto pelos módulos que o *Pesquisador* e o Administrador utilizam, como *Dashboard*, Gráficos, Lista de Alertas e Administração. Como ilustrado, todos esses componentes dependem da validação do componente Auth e Sessão para funcionar, garantindo que nenhum usuário não autenticado acesse os dados (RB05).

Por fim, o *MySQL* é o componente de persistência de dados. O diagrama reforça uma decisão arquitetural importante: o banco de dados nunca é acessado diretamente pelo *Dispositivo* ou pelo *Frontend Web*. O acesso é sempre mediado pelo *Backend PHP*, o que centraliza as regras de negócio e aumenta a segurança do sistema.

4.3.2 Arquitetura de Implantação

Após a definição da arquitetura lógica de componentes (Seção 4.3.1), a arquitetura física e de implantação detalha onde cada componente de *software* é executado e como os nós de *hardware* se comunicam. O Diagrama de Implantação (Figura 4) ilustra essa estrutura física.

Figura 4 – Diagrama de Implantação do Sistema Melissa



Fonte: O autor (2025).

A arquitetura de implantação do sistema é distribuída em três nós físicos principais, como apresentado na Figura 4: o Campo/Laboratório, o Cliente do Usuário e o servidor Hostinger.

O nó Campo/Laboratório representa o protótipo físico da colmeia inteligente. Ele é composto pelo Arduino/Mega, que controla os sensores, e pelo *ESP8266*, que atua como o dispositivo de comunicação. O *ESP8266* se conecta à Rede Wi-Fi local para interagir com o servidor.

O nó Hostinger representa o ambiente de hospedagem na nuvem onde a aplicação *web* reside. Este nó é composto por dois serviços principais. O primeiro é o Servidor *Web*, que hospeda o artefato *Backend PHP*. Este artefato monolítico contém toda a lógica da aplicação, incluindo a *API* de ingestão, o sistema de Auth e o Painel (*frontend*). O segundo serviço é o *MySQL*, o servidor de banco de dados que armazena fisicamente todas as informações, conforme o modelo da Seção 4.2.

O nó Cliente do Usuário representa o dispositivo de acesso do *Pesquisador* ou Administrador. Ele executa o Navegador *Web* (Browser), que é o ambiente onde o Painel (*frontend*) é renderizado.

O diagrama ilustra os dois fluxos de comunicação primários do sistema. O primeiro é o fluxo de ingestão de dados, onde o *ESP8266* envia uma requisição *HTTP* (*POST*) através da Rede Wi-Fi para o Servidor *Web* na Hostinger. O segundo é o fluxo de interação do usuário, onde o Navegador *Web* se comunica com o mesmo Servidor *Web* via *HTTPS* para realizar operações como login, visualização do *dashboard* e gerenciamento. Em ambos os fluxos, o *Backend PHP* é o único componente que processa as requisições e se comunica com o banco de dados *MySQL*.

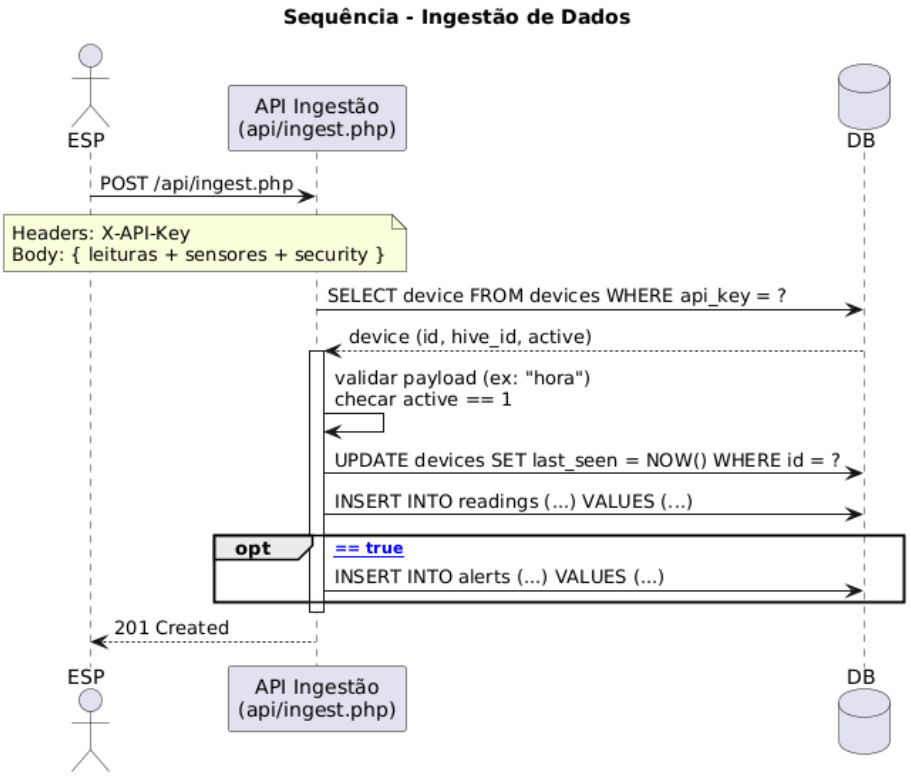
4.4 MODELAGEM COMPORTAMENTAL

4.4.1 Interações de Ingestão de Dados

Para detalhar o comportamento dinâmico do sistema, modelamos o fluxo de interação mais crítico: a ingestão de dados e alertas provenientes do

dispositivo *ESP8266*. O Diagrama de Sequência (Figura 5) ilustra a troca de mensagens passo a passo entre o *ESP* (o dispositivo), a *API* Ingestão (o *backend*) e o *DB* (banco de dados), refletindo a arquitetura de *payload* unificado definida no Documento de Requisitos e implementada nos códigos do sistema.

Figura 5 – Diagrama de Sequência do fluxo de Ingestão de Dados



Fonte: O autor (2025).

A sequência é iniciada pelo Ator *ESP*, que envia uma requisição *POST* para o *endpoint api/ingest.php.* Conforme a nota no diagrama, a requisição inclui o header *X-API-Key* (RB01) e um corpo *JSON* contendo o *payload* unificado com todas as leituras dos sensores e o status de segurança ({leituras + sensores + *security*}).

Ao receber a requisição, a *API* Ingestão primeiro consulta o DB para validar a *X-API-Key* e verificar se o dispositivo está ativo (*checar active == 1*), atendendo às regras de negócio RB02 e RB04. Após validar o dispositivo e o *payload* (incluindo a formatação da data, como implementado no *ingest.php*), a

API executa as operações de escrita no DB. Primeiro, atualiza o campo *last_seen* na tabela devices (RF06). Em seguida, insere os dados dos sensores na tabela readings (RF04).

Posteriormente, a *API* Ingestão avalia o status de segurança recebido no *payload*. Conforme modelado pelo fragmento opcional (opt) com a condição *[== true],* apenas se o campo *security.opened* for verdadeiro, a *API* executa um INSERT adicional na tabela alerts (RF05, RB03).

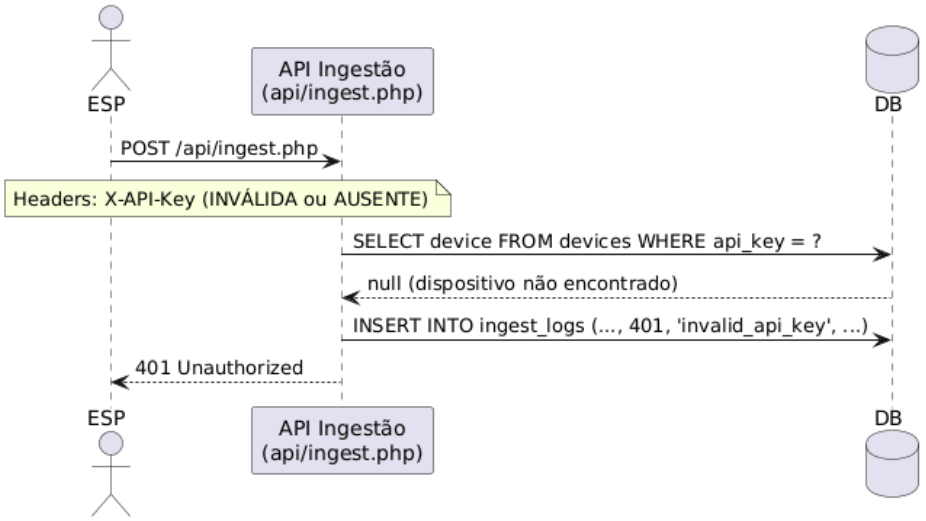
Finalmente, tendo concluído com sucesso as operações no banco de dados (seja apenas a leitura ou a leitura mais o alerta), a API Ingestão retorna uma resposta 201 Created para o ESP, indicando que os dados foram recebidos e processados.

Esta modelagem detalha a implementação do fluxo principal do sistema, validando a interação entre o *hardware* e o *backend* e demonstrando o cumprimento dos requisitos funcionais e regras de negócio relacionadas à ingestão de dados.

4.4.2 Interação em Falhas de Autenticação

Além do fluxo de sucesso na ingestão de dados, é crucial modelar como o sistema lida com falhas, especialmente as de autenticação. O Diagrama de Sequência (Figura 6) ilustra o comportamento do sistema quando o dispositivo ESP tenta enviar dados com uma *X-API-Key* inválida ou ausente, conforme a regra de negócio RB02 e os requisitos de log RF10 e RNF07.

Figura 6 – Diagrama de Sequência do fluxo de Falha de Autenticação (API Key Inválida)



Fonte: O autor (2025).

A sequência é iniciada pelo Ator *ESP* enviando a requisição *POST* para *api/ingest.php*. A nota no diagrama indica que, neste cenário, o header *X-API-Key* está inválido ou ausente.

A *API* Ingestão recebe a requisição e, como primeiro passo de validação, consulta o *DB* para encontrar um dispositivo correspondente à *API* Key fornecida (*SELECT device* ...). Como a chave é inválida, o DB retorna null (dispositivo não encontrado) para a *API*.

Ao detectar que nenhum dispositivo válido foi encontrado, a *API* Ingestão interrompe o fluxo de processamento normal. Antes de responder ao ESP, ela executa uma operação de escrita no DB: um INSERT INTO *ingest_logs*. Este registro de log armazena detalhes da tentativa falha, incluindo o status 401 e a mensagem de erro '*invalid_api_key*', cumprindo os requisitos de monitoramento e auditoria RF10 e RNF07.

Finalmente, a *API* Ingestão retorna a resposta 401 Unauthorized para o ESP, indicando que a requisição foi rejeitada por falta de autenticação válida, conforme especificado na regra de negócio RB02.

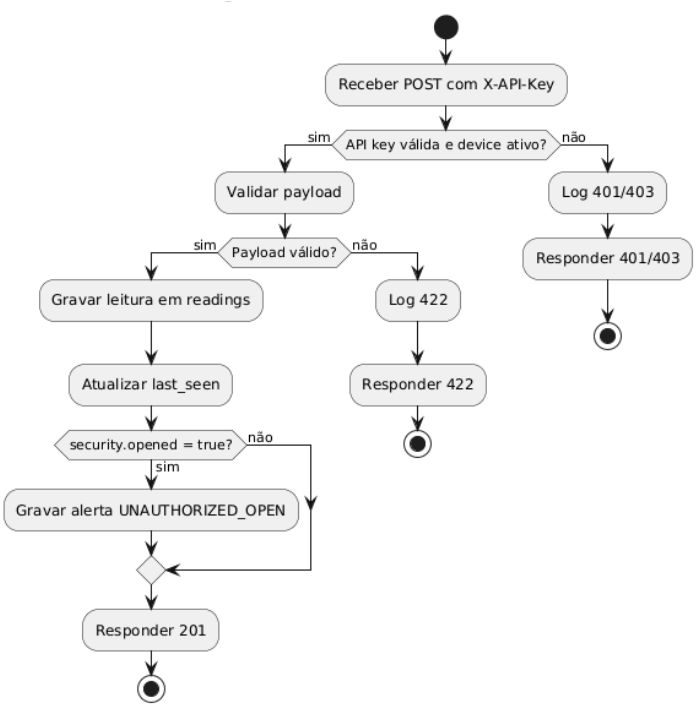
Esta modelagem demonstra o tratamento de erro implementado no *backend*, garantindo que apenas dispositivos autenticados possam enviar

dados e que todas as tentativas inválidas sejam devidamente registradas para análise posterior.

4.4.3 Fluxo de Atividade

Para complementar a visão dinâmica das interações, utilizou-se um *Diagrama de Atividade UML* para modelar o fluxo de controle interno do processo mais crítico do sistema: a ingestão de dados realizada pelo componente *API* Ingestão (*api/ingest.php*). O diagrama (Figura 7) detalha a sequência de passos e as decisões lógicas executadas pelo *backend* ao receber uma requisição do dispositivo *ESP8266*.

Figura 7 – Diagrama de Atividade do Fluxo de Ingestão de Dados



Fonte: O autor (2025).

O fluxo inicia com a ação Receber POST com *X-API-Key*, representando a chegada da requisição *HTTP* (RF03). A primeira decisão (*API ke*y válida e device ativo?) verifica a autenticidade e a autorização do dispositivo, consultando o banco de dados conforme as regras RB01, RB02 e RB04 . Em caso de falha (não), o sistema executa as ações Log 401/403 (RF10, RNF07) e Responder 401/403, terminando o fluxo imediatamente ao direcionar para o nó final, refletindo o comportamento da função reject().

Se a autenticação for bem-sucedida (sim), o fluxo prossegue para a ação Validar *payload*, onde o *backend* verifica a estrutura e o formato dos dados recebidos (como a data/hora). A decisão seguinte (*Payload* válido?) direciona o fluxo: em caso de falha (não), as ações *Log 422* (RF10, RNF07) e Responder 422 são executadas, terminando o fluxo imediatamente ao direcionar para o nó final.

Se o *payload* for válido (sim), o sistema executa as ações principais de persistência: Gravar leitura em readings (RF04) e Atualizar *last_seen* (RF06). Em seguida, a decisão (*security.opened = true?*) verifica o status de segurança reportado no *payload* (RB03). Se a condição for verdadeira (sim), a ação Gravar alerta *UNAUTHORIZED_OPEN* (RF05) é executada. Caso a condição seja falsa (não), esta ação é contornada. Ambos os caminhos (com ou sem gravação de alerta) convergem em um ponto de junção antes de prosseguir para a ação final Responder 201, que sinaliza o sucesso da operação para o dispositivo *ESP8266* e encerra o fluxo no nó final.

Este diagrama detalha a lógica implementada no *backend* (ingest.php), assegurando que todos os requisitos funcionais, não funcionais e regras de negócio pertinentes ao fluxo de ingestão, incluindo o tratamento de erros e o fluxo condicional de alertas, sejam corretamente contemplados.

4.5 DETALHES DA IMPLEMENTAÇÃO

Com as tecnologias já justificadas na Seção 3.1, passamos a detalhar a implementação dos componentes críticos do sistema. Focamos em três áreas principais: a *API* de ingestão, a segurança dos usuários e a visualização dos dados.

O principal ponto de entrada para o dispositivo é a *API* de Ingestão. Ela foi implementada como um único script, *api/ingest.php*, o que centraliza o recebimento de dados. A implementação segue o fluxo do Diagrama de Atividade (Figura 7). A requisição é lida usando *php://input*, e a primeira etapa é sempre a validação da *X-API-Key* (RB02), que envolve uma consulta *SELECT* no banco.

Um aspecto importante da implementação, visando a integridade dos dados (RNF04), é o uso de transações SQL. O script inicia uma transação com *pdo->beginTransaction()* antes de qualquer escrita. Isso garante que as operações (*UPDATE devices.last_seen* (RF06) , *INSERT INTO readings* (RF04) e o *INSERT* condicional em *alerts*) sejam atômicas.

Se qualquer erro ocorrer durante esse processo, o *pdo->rollBack()* é acionado, desfazendo todas as operações. Isso garante que o banco de dados não entre em um estado inconsistente.

Para a Segurança e Autenticação de Usuários, o foco foi atender aos requisitos RF07 , RNF03 e RB05. Implementamos um sistema de autenticação baseado em Sessões *PHP*. As senhas dos usuários, conforme o RNF03, não são armazenadas em texto claro; elas são protegidas no banco usando a função password_hash() nativa do *PHP*.

No fluxo de login (Autenticar usuário), o *script* de autenticação compara a senha fornecida com o hash armazenado usando a função *password_verify()*. Em caso de sucesso, as credenciais do usuário (como id e role) são armazenadas na *$_SESSION* do *PHP*. Todas as páginas protegidas (como o Dashboard e a Administração) verificam no início se a sessão é válida, redirecionando para o login caso contrário, o que cumpre a regra de negócio RB05.

Visualização de Dados (*Dashboard* e Gráficos) A implementação do *Dashboard* e dos Gráficos (RF08) envolve uma colaboração entre o *backend PHP* e o *frontend JavaScript*. O *script PHP*, no lado do servidor, é responsável por executar as consultas *SQL* no *MySQL*. Ele busca, por exemplo, as últimas N leituras para os cartões de "tempo real" e um conjunto de dados históricos (ex: últimas 24 horas) para os gráficos.

Esses dados são então formatados pelo *PHP* (geralmente em um formato *JSON* embutido na própria página HTML) e entregues ao cliente. No navegador, a biblioteca *Chart.js* é inicializada via *JavaScript*. Ela lê esse *JSON* de dados e renderiza os gráficos interativos de temperatura, umidade e peso, permitindo que o *Pesquisador* analise o histórico das variáveis ambientais.

4.6 ESTRATÉGIA DE TESTES E VALIDAÇÃO

A etapa final da metodologia *DSR* (*Design Science Research*) é a avaliação do artefato. Para validar o Sistema *Web* Melissa, foi definida uma estratégia de testes em três frentes: testes de *API* (caixa-preta), testes de integração ponta-a-ponta e testes funcionais da interface (*frontend*). O objetivo foi verificar o cumprimento de todos os requisitos funcionais, não funcionais e regras de negócio definidos no Apêndice A .

A validação inicial do *backend* (*api/ingest.php*) foi realizada por meio de testes de *API* (caixa-preta), simulando as requisições do dispositivo com uma ferramenta de cliente HTTP (Postman). Testamos o cenário de sucesso enviando um POST com uma *X-API-Key* válida e um *payload JSON* completo . Verificou-se se o sistema retornou 201 Created e se os dados foram persistidos corretamente nas tabelas readings e devices, conforme o fluxo da Figura 7. O fluxo de alerta também foi validado enviando um *payload* com *security.opened* definido como *true*, o que deveria criar um registro na tabela alerts (RF05, RB03). Adicionalmente, os cenários de exceção foram testados conforme a Figura 6, enviando requisições com *X-API-Key* inválida (esperando *401 Unauthorized*), dispositivo inativo (esperando *403 Forbidden*) e *JSON* mal formatado (esperando *422 Unprocessable Entity*), verificando se todas as falhas foram registradas na tabela *ingest_logs* (RF10, RNF07).

Para validar o sistema em condições reais de operação, realizamos testes de integração ponta-a-ponta (end-to-end). Nesta fase, conectamos o protótipo de *hardware* (Arduino + *ESP8266*) à rede Wi-Fi, permitindo que ele enviasse dados reais para o ambiente de produção na Hostinger. Em paralelo, acessamos o *Dashboard* (RF08) no navegador. Validamos que as leituras (temperatura, umidade, luminosidade, etc.) que apareciam na interface refletiam os dados enviados pelo protótipo. O teste de segurança (RF05, RB03) foi o mais importante: disparamos fisicamente o sensor de abertura no protótipo e confirmamos que o alerta *UNAUTHORIZED_OPEN* foi exibido na Lista de Alertas da interface *web* em poucos segundos.

Por fim, os testes funcionais da interface (*frontend*) foram realizados manualmente, baseados nos casos de uso (Figura 1). A autenticação e o

controle de acesso (RF07, RB05) foram validados ao tentar acessar rotas protegidas sem login e ao verificar as permissões distintas entre os perfis *Pesquisador* e Administrador. As funcionalidades principais, como a exibição de dados no *Dashboard*, a renderização dos Gráficos (RF08) e a Exportação CSV (RF09), foram verificadas quanto ao seu funcionamento correto. O requisito de responsividade (RNF05) foi validado utilizando as ferramentas de desenvolvedor do navegador para simular o acesso em diferentes tamanhos de tela.

Todos os testes foram executados com o objetivo de validar as métricas de avaliação definidas (Apêndice A, Seção 11) . A latência da *API* (RNF02) foi aferida com o Postman; a taxa de detecção de abertura (100%) foi validada no teste ponta-a-ponta; e a taxa de ingestão bem-sucedida (> 99,5%) foi avaliada por meio de um script de teste de carga.

5. RESULTADOS E DISCUSSÃO

A finalidade desta seção é demonstrar como o protótipo da interface *web* atende, na prática, aos objetivos especificados. Serão apresentadas as principais telas do Sistema Melissa, conectando suas funcionalidades diretamente aos requisitos funcionais (RF) e regras de negócio (RB) estabelecidas no Documento de Requisitos (Apêndice A).

5.1 APRESENTAÇÃO DA INTERFACE WEB

A interface *web* é o principal ponto de interação entre os atores humanos (*Pesquisador* e *Administrador*) e os dados coletados pelo dispositivo. O seu desenvolvimento focou na responsividade (RNF05) e na clareza da informação.

A seguir, apresentamos as telas que compõem o fluxo de visualização e gerenciamento do sistema (Figura 8).

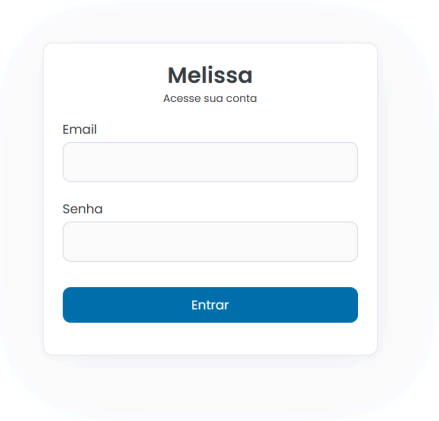
5.1.1 Autenticação e Controle de Acesso

O acesso ao Sistema Melissa é a primeira barreira de proteção e controle, implementando diretamente os requisitos de segurança da informação. A interface não permite qualquer visualização de dados sem a devida autenticação, em conformidade estrita com a regra de negócio RB05

"Apenas usuários autenticados podem acessar *dashboards* e relatórios" e o requisito funcional RF07 "Autenticar usuários por e-mail e senha".

A tela de login, apresentada na Figura 8, é a porta de entrada do sistema.

Figura 8 – Tela de Autenticação do Sistema Melissa



Fonte: O autor (2025).

Como demonstrado na figura, a tela foi projetada com uma estética minimalista, alinhada à escolha do framework PicoCSS, garantindo seu funcionamento adequado em dispositivos móveis e atendendo ao requisito de responsividade (RNF05). Conforme detalhado na implementação, o *backend* valida as credenciais fornecidas (Email e Senha) contra os hashes seguros armazenados na tabela users do banco de dados, atendendo ao requisito RNF03.

Ao autenticar-se com sucesso, o usuário é direcionado ao painel principal do sistema.

5.1.2 Painel de Controle (Dashboard)

O *Dashboard*, ou Painel de Controle, é a tela central do Sistema Melissa e o principal instrumento de trabalho do ator *Pesquisador*. Esta tela foi projetada para consolidar e apresentar, de forma clara e acessível, o volume de

dados coletados, atendendo diretamente ao requisito RF08 "Exibir *dashboard* com últimas leituras, gráficos de variáveis ambientais e lista de alertas".

As Figuras 9 e 10 ilustram os componentes desta tela.

Figura 9 – Dashboard do Sistema Melissa (Filtros, Leituras e Alertas)

Fonte: O autor (2025).

Como observado na Figura 9, o topo da tela apresenta os controles de visualização. O usuário pode selecionar a colmeia que deseja analisar (neste caso, 'HIVE-01') e definir um período de tempo (Início e Fim) para a consulta. Adjacente a estes filtros, encontram-se os botões "Exportar Leituras (CSV)" e "Exportar Alertas (CSV)", que implementam diretamente o requisito funcional RF09 – "Filtrar e exportar dados e alertas em CSV".

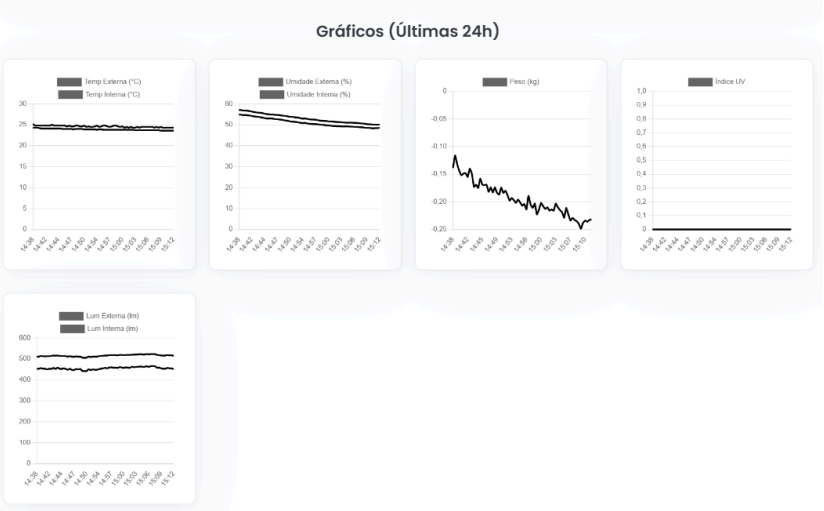
Abaixo dos filtros, a tela se divide em dois painéis principais:

1. Última Leitura: Este painel fornece ao pesquisador a visão mais imediata da condição da colmeia. Ele exibe o timestamp da última comunicação bem-sucedida ("Atualizado em: 04/11/2025 às 15:12"), uma funcionalidade que depende da atualização do campo last_seen (RF06) no banco de dados. O painel consolida todas as variáveis ambientais coletadas (Temperatura, Umidade, Luminosidade, Peso e UV) , persistidas na tabela readings (RF04).
2. Alertas Recentes: Este é o componente de segurança do sistema. Sua função é exibir, de forma prioritária, qualquer evento de violação detectado. Na captura de tela, o sistema informa "Nenhum alerta

recente", indicando um estado de normalidade operacional. É neste painel que um alerta do tipo *UNAUTHORIZED_OPEN* seria exibido, caso o *backend* o registrasse (RF05 , RB03).

A segunda parte do *dashboard* é dedicada à análise histórica, conforme detalhado na Figura 10.

Figura 10 – Dashboard do Sistema Melissa (Gráficos Históricos)



Fonte: O autor (2025).

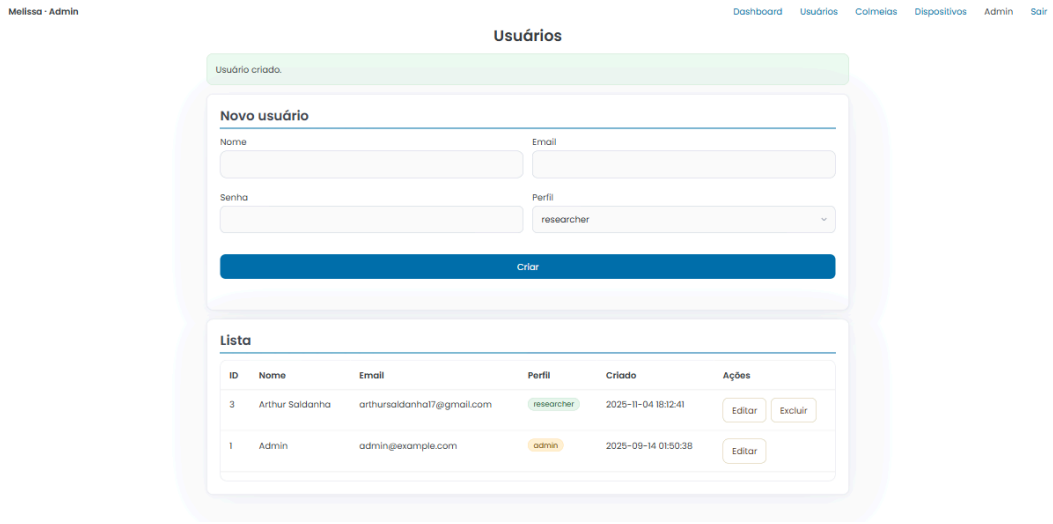
Esta seção implementa a exibição de "gráficos de variáveis ambientais" (parte do RF08). Utilizando a biblioteca Chart.js (justificada na Seção 3.1), a interface renderiza o histórico das últimas 24 horas para as principais variáveis. A Figura 10 demonstra a visualização dos gráficos de Temperatura (Interna e Externa), Umidade (Interna e Externa), Peso, Índice UV e Luminosidade (Interna e Externa) . Essa visualização histórica é muito importante para que o pesquisador possa identificar tendências, padrões de comportamento ou anomalias que não seriam perceptíveis analisando apenas a última leitura.

5.1.3 Gerenciamento do Sistema

Enquanto o *dashboard* é o foco do ator *Pesquisador*, o Sistema Melissa possui uma área administrativa dedicada, acessível apenas a usuários com perfil de 'admin', conforme definido nos atores. Esta área implementa os casos de uso de gerenciamento (Figura 1) e permite a configuração da base do sistema.

A Figura 11 apresenta a tela de Gerenciamento de Usuários.

Figura 11 – Tela de Gerenciamento de Usuários



Fonte: O autor (2025).

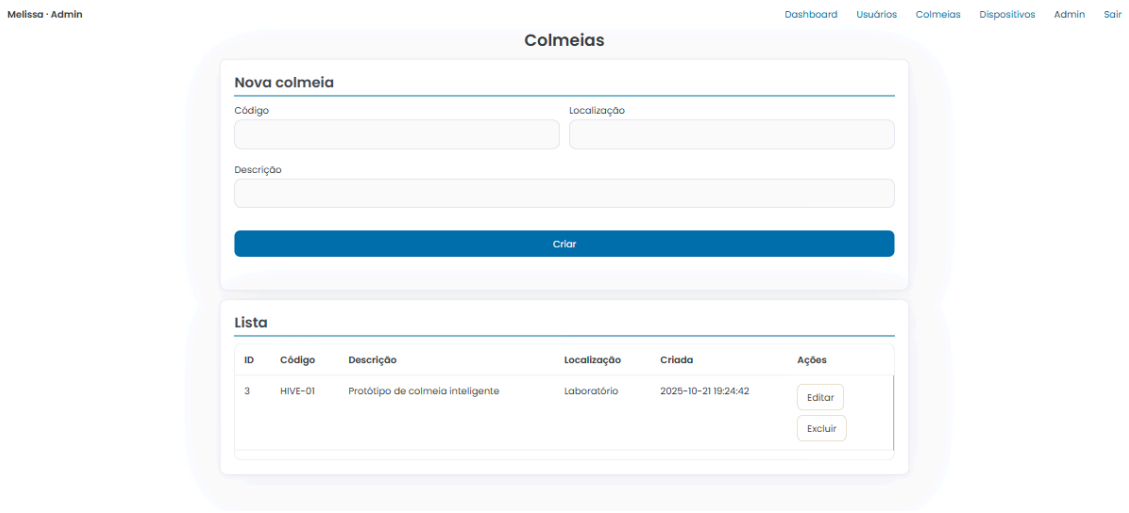
Esta tela é a implementação prática da gestão de contas de acesso (RF07 – "Autenticar usuários por e-mail e senha"). Nela, o Administrador pode executar o ciclo completo de gerenciamento de contas. O formulário "Novo usuário" permite o cadastro de novas contas, exigindo Nome, Email, Senha e a definição de um Perfil. Este campo "Perfil", com as opções 'researcher' e 'admin', implementa o controle de acesso baseado em papéis (roles), conforme modelado nos Atores e na tabela users. Complementando a criação, a seção "Lista" exibe todos os usuários cadastrados no sistema, permitindo ao administrador editar ou excluir contas existentes, o que completa a gestão de autenticação.

Esta funcionalidade é fundamental para o ciclo de vida da aplicação, permitindo que o administrador conceda ou revogue o acesso de pesquisadores ao sistema.

5.1.4 Gerenciamento de Colmeias e Dispositivos

Complementando a gestão de usuários, a área administrativa permite o gerenciamento dos componentes físicos (ou lógicos) do sistema: as colmeias e os dispositivos. A Figura 12 ilustra a interface de gerenciamento de colmeias.

Figura 12 – Tela de Gerenciamento de Colmeias

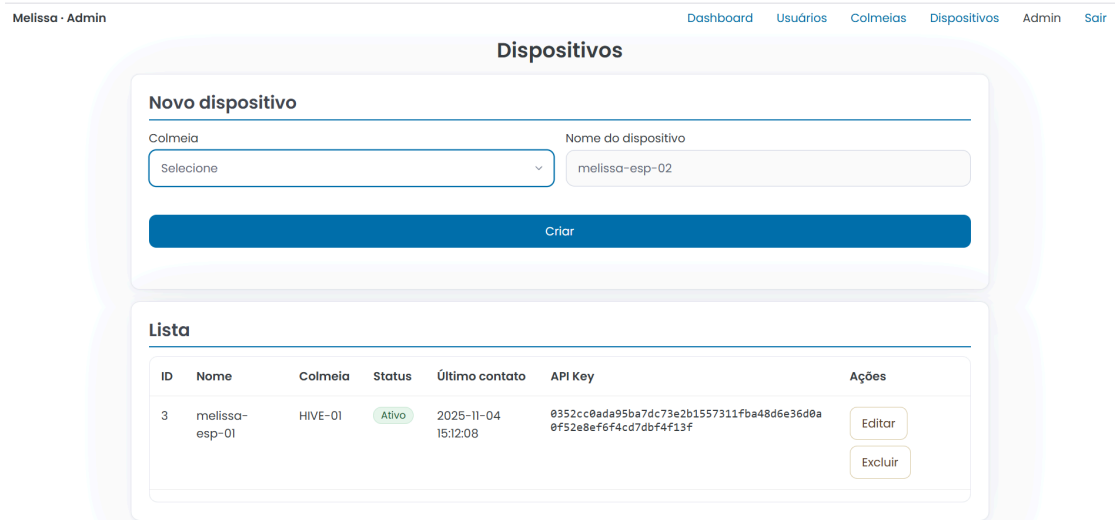


Fonte: O autor (2025).

Esta tela implementa diretamente o requisito RF01 – "Cadastrar colmeias com código, descrição e localização". O formulário "Nova colmeia" permite ao Administrador registrar uma nova colmeia física no sistema, associando a ela um Código (ex: 'HIVE-01'), uma Descrição (ex: 'Protótipo de colmeia inteligente') e sua Localização (ex: 'Laboratório'). A "Lista" exibe os registros existentes na tabela hives, permitindo sua edição ou exclusão.

Este cadastro é a etapa inicial para a configuração de um novo ponto de monitoramento. Após o registro da colmeia, o administrador pode então vincular o módulo de *hardware* (*ESP8266*) a este registro.
A Figura 13 apresenta a tela de Gerenciamento de *Dispositivos*, que conclui este fluxo de configuração.

Figura 13 – Tela de Gerenciamento de Dispositivos



Fonte: O autor (2025).

Esta interface é a implementação direta do RF02 – "Cadastrar dispositivos vinculados a colmeias, gerando *API key* única". No formulário "Novo dispositivo", o Administrador seleciona a "Colmeia" (criada na etapa anterior, Figura 11) e define um nome para o *hardware* (ex: 'melissa-esp-02'), estabelecendo o relacionamento modelado na Seção 4.2.

A "Lista" de dispositivos é onde os resultados mais importantes desta etapa são visualizados. A coluna "*API Key*" exibe o identificador único gerado pelo sistema (ex: 0352cc...), que cumpre a RB01 – "Cada dispositivo possui uma *API key* única e obrigatória". Esta chave é o token de autenticação que deve ser gravado no *ESP8266* para permitir o envio de dados. Além disso, a lista exibe o "Status" do dispositivo ('Ativo'), que é a condição verificada pela *API* para aceitar leituras (RB04) , e o "Último contato", que é o timestamp da última ingestão bem-sucedida, atendendo ao RF06 – "Atualizar automaticamente o campo last_seen".

Com a apresentação destas telas administrativas (Usuários, Colmeias e *Dispositivos*), o ciclo de configuração e uso do sistema pela interface *web* está demonstrado. A Seção 5.1 validou visualmente o artefato. A próxima seção (5.2) focará em validar o comportamento do sistema, apresentando os resultados dos testes de *API* e integração ponta-a-ponta.

5.2 VALIDAÇÃO DOS TESTES E ADERÊNCIA AOS REQUISITOS

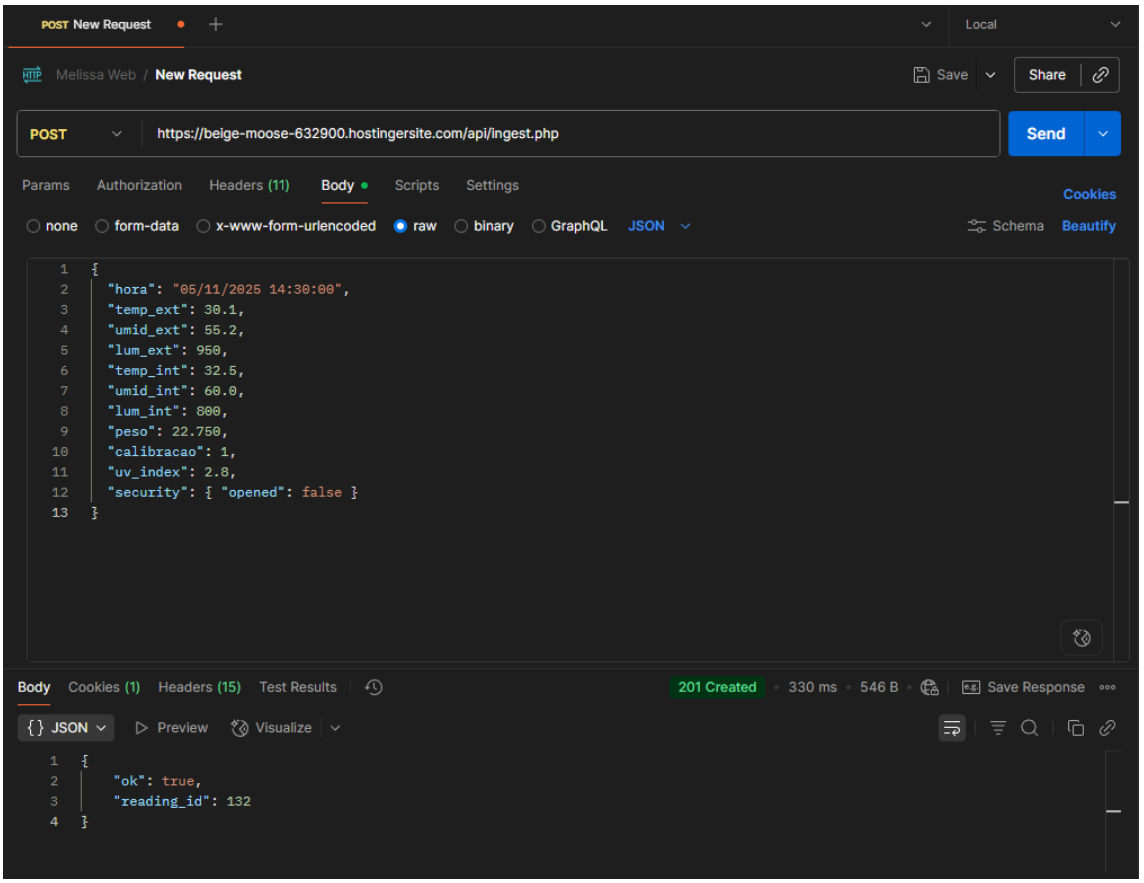
Após a apresentação da interface *web* (Seção 5.1), esta seção foca em demonstrar o comportamento do sistema e validar sua aderência aos requisitos e regras de negócio. Conforme a estratégia definida na Seção 4.6, a validação foi dividida em frentes de teste, começando pela avaliação do *backend* (*API* de ingestão) e pela integração ponta-a-ponta.

5.2.1 Testes de API (Caixa-Preta) e Integração Ponta-a-Ponta

A validação do *backend* (*api/ingest.php*) foi realizada por meio de testes de *API* (*caixa-preta*), simulando as requisições do dispositivo *ESP8266* com um cliente *HTTP (Postman)*.

O primeiro teste visou validar o fluxo principal de ingestão de dados. Foi simulado um envio de *payload* completo, com uma *X-API-Key* válida. A Figura 14 apresenta o resultado bem-sucedido desta requisição.

Figura 14 – Teste de API: Sucesso na Ingestão (Postman)



Fonte: O autor (2025).

O sistema respondeu com “201 Created”, confirmando que o requisito RF03 (Receber dados do *ESP8266*) foi atendido. A resposta JSON {"ok": *true*, "reading_id": 132} demonstra que a persistência no banco (RF04) ocorreu com sucesso. Adicionalmente, o tempo de resposta de 330 ms atende com folga ao requisito não funcional RNF02, que estipulava uma latência de ingestão abaixo de 500 ms (p95).

O teste mais crítico foi a validação da integração entre o *payload* do dispositivo e a interface de alertas. Para isso, uma segunda requisição foi enviada via Postman, desta vez com o campo de segurança alterado para {"*security*": { "*opened*": *true* }}. O sistema respondeu com “201 *Created*” indicando o sucesso da ingestão e, como resultado, gerou o alerta de segurança, conforme a RB03 ("Se *security.opened* = *true*, o sistema gera automaticamente um alerta *CRITICAL*").

A Figura 15 captura o resultado deste teste, visualizado diretamente no *Dashboard* do sistema.

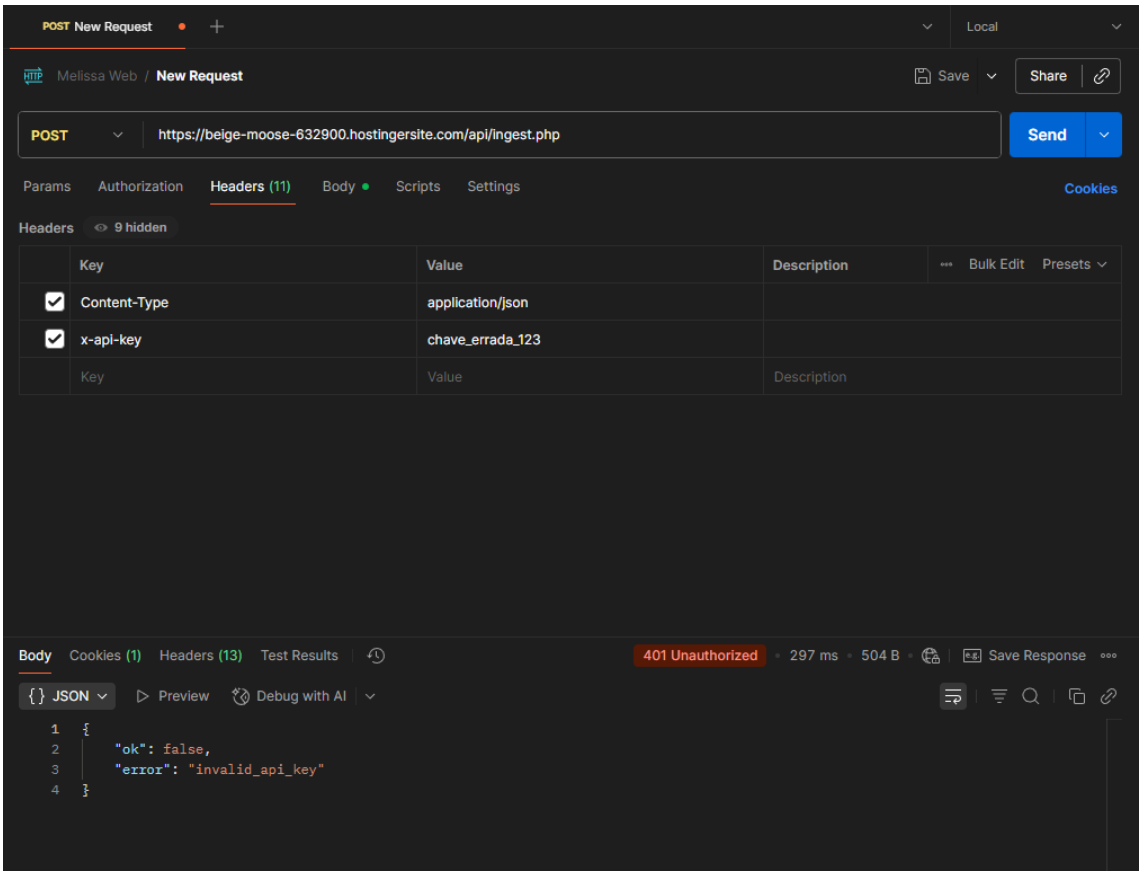
Figura 15 – Teste de Integração: Alerta de Segurança no Dashboard

Fonte: O autor (2025).

Esta tela valida o fluxo de segurança ponta-a-ponta. O *payload* enviado (Temp: 30.1°C, Peso: 22.75 kg, etc.) foi corretamente persistido (RF04) e exibido em "Última Leitura". Mais importante, o painel "Alertas Recentes" exibe "Abertura não autorizada detectada", comprovando que o *backend* implementou a RB03, registrou o alerta (RF05), e o *frontend* o exibiu corretamente para o usuário, cumprindo o RF08.

Para completar a validação da *API*, foram testados os cenários de falha. A Figura 16 demonstra a tentativa de envio de dados utilizando uma *x-api-key* inválida.

Figura 16 – Teste de API: Falha de Autenticação (Postman)



Fonte: O autor (2025).

O resultado comprova a implementação da RB02 ("Requisições sem *X-API-Key* válida devem ser rejeitadas (401)"). O servidor rejeitou a requisição com o status *401 Unauthorized* e retornou a mensagem de erro *invalid_api_key*, conforme modelado no Diagrama de Sequência de Falha (Figura 6).

Finalmente, foi verificado se esta falha foi devidamente registrada, conforme exigido pelos requisitos de auditoria RF10 e RNF07.

Figura 17 – Teste de API: Log da Falha de Autenticação (Banco de Dados)

Fonte: O autor (2025).

A Figura 17 evidencia que a tabela *ingest_logs* registrou a tentativa de falha. O log armazenou o *status_code (401)*, a *error_message* (*invalid_api_key*) e o *payload* rejeitado, cumprindo integralmente os requisitos de log. O campo *device_id* está corretamente registrado como *NULL*, pois o sistema não pôde identificar o dispositivo devido à chave inválida.

5.2.2 Testes Funcionais da Interface

A segunda frente de validação focou nos testes manuais da interface *web*, verificando os requisitos de acesso e usabilidade, conforme a estratégia da Seção 4.6.

O primeiro teste funcional visou validar o controle de acesso baseado em papéis, que restringe o acesso a usuários autenticados (RB05) e separa as permissões dos atores. A Figura 18 demonstra a interface do sistema quando acessada por um usuário com perfil de "*Pesquisador*".

Figura 18 – Teste Funcional: Visão do Ator "*Pesquisador*"

Fonte: O autor (2025).

A tela comprova que o usuário "*Pesquisador*" pode se autenticar (RF07) e acessar o *Dashboard* para visualizar dados e alertas, conforme seu caso de uso (Figura 1). Crucialmente, o menu de navegação superior não exibe os links "Usuários", "Colmeias" ou "*Dispositivos*". Isso confirma que as rotas administrativas estão protegidas e são inacessíveis a este perfil, implementando corretamente a separação de privilégios entre os atores "*Pesquisador*" e "*Administrador*".

O teste final validou o requisito não funcional de usabilidade RNF05 – "Interface *web* deve ser responsiva (desktop e mobile)". A interface foi inspecionada em um simulador de dispositivo móvel, como apresentado na Figura 19.

Figura 19 – Teste Funcional: Responsividade da Interface em Visão Mobile

Fonte: O autor (2025).

As capturas de tela demonstram que, em uma tela estreita, o layout do *dashboard* se adapta. Os componentes, como "Última Leitura", "Alertas Recentes" e "Gráficos", que são exibidos lado a lado em um desktop, são aqui empilhados verticalmente. Esta adaptação confirma que o requisito RNF05 foi

atendido, garantindo que os usuários possam acessar e consumir os dados de forma clara em smartphones ou tablets.

O teste funcional final validou a capacidade de exportação de dados do sistema, um requisito-chave para o ator *Pesquisador*. A Figura 20 mostra o resultado da utilização da funcionalidade "Exportar Leituras (CSV)".

Figura 20 – Teste Funcional: Resultado da Exportação de Dados (CSV)

	A	B	C	D	E	F	G	H	I	J	K	L
1	ts	calibracao	hora	lum_ext	lum_int	peso	raw_json	temp_ext	temp_int	umid_ext	umid_int	uv_index
2	2025-11-04 14:38:55	999999	2025-11-04 11:36:07	511	452	-139	{"hora":"04/11/2025 11:36:07","temp_ext":24.3,"u	24.30	25.20	54.90	57.10	0.00
3	2025-11-04 14:39:25	999999	2025-11-04 11:36:37	511	453	-116	{"hora":"04/11/2025 11:36:37","temp_ext":24.3,"u	24.30	24.80	54.80	57.00	0.00
4	2025-11-04 14:39:55	999999	2025-11-04 11:37:07	514	456	-132	{"hora":"04/11/2025 11:37:07","temp_ext":24.3,"u	24.30	24.80	54.60	56.80	0.00
5	2025-11-04 14:40:25	999999	2025-11-04 11:37:37	513	453	-144	{"hora":"04/11/2025 11:37:37","temp_ext":24.2,"u	24.20	24.80	54.70	56.80	0.00
6	2025-11-04 14:40:55	999999	2025-11-04 11:38:07	512	453	-152	{"hora":"04/11/2025 11:38:07","temp_ext":24.1,"u	24.10	24.80	54.50	56.70	0.00
7	2025-11-04 14:41:26	999999	2025-11-04 11:38:38	513	450	-149	{"hora":"04/11/2025 11:38:38","temp_ext":24.1,"u	24.10	24.80	54.40	56.50	0.00
8	2025-11-04 14:42:00	999999	2025-11-04 11:39:08	513	453	-148	{"hora":"04/11/2025 11:39:08","temp_ext":24.1,"u	24.10	24.80	54.20	56.30	0.00
9	2025-11-04 14:42:26	999999	2025-11-04 11:39:38	514	452	-155	{"hora":"04/11/2025 11:39:38","temp_ext":24.1,"u	24.10	24.80	54.00	56.10	0.00
10	2025-11-04 14:42:56	999999	2025-11-04 11:40:08	516	457	-140	{"hora":"04/11/2025 11:40:08","temp_ext":24.1,"u	24.10	24.80	53.90	55.90	0.00
11	2025-11-04 14:43:26	999999	2025-11-04 11:40:38	515	452	-149	{"hora":"04/11/2025 11:40:38","temp_ext":24.1,"u	24.10	25.00	53.80	55.80	0.00
12	2025-11-04 14:43:56	999999	2025-11-04 11:41:09	516	458	-173	{"hora":"04/11/2025 11:41:09","temp_ext":24.1,"u	24.10	24.80	53.60	55.70	0.00
13	2025-11-04 14:44:26	999999	2025-11-04 11:41:39	514	453	-169	{"hora":"04/11/2025 11:41:39","temp_ext":24.1,"u	24.10	24.80	53.40	55.60	0.00
14	2025-11-04 14:44:57	999999	2025-11-04 11:42:09	514	452	-175	{"hora":"04/11/2025 11:42:09","temp_ext":24.1,"u	24.10	24.80	53.20	55.30	0.00
15	2025-11-04 14:45:27	999999	2025-11-04 11:42:39	514	455	-158	{"hora":"04/11/2025 11:42:39","temp_ext":24.1,"u	24.10	24.80	53.00	55.10	0.00
16	2025-11-04 14:45:57	999999	2025-11-04 11:43:09	513	452	-169	{"hora":"04/11/2025 11:43:09","temp_ext":24,"um	24.00	24.80	53.00	55.00	0.00
17	2025-11-04 14:46:27	999999	2025-11-04 11:43:40	511	448	-170	{"hora":"04/11/2025 11:43:40","temp_ext":24,"um	24.00	24.80	53.00	54.90	0.00
18	2025-11-04 14:46:58	999999	2025-11-04 11:44:10	513	453	-169	{"hora":"04/11/2025 11:44:10","temp_ext":24,"um	24.00	24.60	52.90	54.90	0.00
19	2025-11-04 14:47:28	999999	2025-11-04 11:44:40	511	447	-182	{"hora":"04/11/2025 11:44:40","temp_ext":24,"um	24.00	24.80	52.70	54.70	0.00
20	2025-11-04 14:47:58	999999	2025-11-04 11:45:10	510	447	-174	{"hora":"04/11/2025 11:45:10","temp_ext":24,"um	24.00	24.60	52.70	54.70	0.00
21	2025-11-04 14:48:28	999999	2025-11-04 11:45:40	512	452	-183	{"hora":"04/11/2025 11:45:40","temp_ext":23.9,"u	23.90	24.60	52.50	54.60	0.00
22	2025-11-04 14:48:58	999999	2025-11-04 11:46:11	511	450	-174	{"hora":"04/11/2025 11:46:11","temp_ext":24,"um	24.00	24.80	52.40	54.40	0.00
23	2025-11-04 14:49:28	999999	2025-11-04 11:46:41	511	452	-184	{"hora":"04/11/2025 11:46:41","temp_ext":24,"um	24.00	24.80	52.20	54.30	0.00

Fonte: O autor (2025).

A imagem comprova que o sistema gerou com sucesso um arquivo .csv contendo os dados históricos de leituras. O arquivo está corretamente formatado, com timestamps e todas as variáveis ambientais (temp_ext, temp_int, umid_ext, peso, uv_index, etc.) separadas em colunas, prontas para análise em *softwares* externos (como Excel ou Google Sheets). Este resultado cumpre integralmente o requisito RF09 – "Filtrar e exportar dados e alertas em CSV".

5.3 DISCUSSÃO DOS RESULTADOS

Do ponto de vista metodológico, uma das principais dificuldades e, ao mesmo tempo, um achado relevante esteve na estratégia de validação. O planejamento previa testes de integração ponta-a-ponta conectando o hardware (Arduino + *ESP8266*) ao sistema. No entanto, durante a fase de desenvolvimento deste TCC, o protótipo físico encontrava-se em um processo paralelo de manutenção e calibração de sensores. Diante disso, optou-se por uma abordagem metodológica rigorosa de testes de API (caixa-preta), utilizando a ferramenta *Postman* para simular com precisão o comportamento e os *payloads* do dispositivo *ESP8266*. Essa decisão, embora motivada por uma limitação, permitiu um ambiente de teste controlado e isolado. A estratégia adotada demonstra que o software atende aos requisitos especificados e está tecnicamente pronto para a integração com o hardware em campo, com uma *API* de ingestão segura e aderente às regras de negócio.

À luz dos trabalhos correlatos apresentados na Seção 1.6, os resultados obtidos com o Sistema Melissa permitem posicionar este artefato dentro do contexto da Apicultura de Precisão baseada em IoT. Estudos como o de SILVA (2017) enfatizam o monitoramento não invasivo das colmeias, enquanto ZACEPINS et al. (2020) discutem a Apicultura de Precisão em três grandes fases, coleta, análise e aplicação de dados, destacando a necessidade de sistemas que transformem medições em decisões práticas. Já trabalhos recentes, como os de CARDOSO et al. (2023) e MARQUES et al. (2024), demonstram o uso de protótipos de baixo custo e arquiteturas IoT para coleta de temperatura, umidade, peso e outros parâmetros ambientais, reforçando a viabilidade técnica de monitorar colônias individuais em tempo quase real.

Os testes apresentados nas Figuras 14 a 20 evidenciam que o Sistema Melissa avança principalmente nas etapas de coleta e apresentação de dados, em consonância com essa literatura, mas também adiciona elementos que não aparecem de forma integrada nos trabalhos correlatos. A API de ingestão validada por testes de caixa-preta atende aos requisitos de latência, autenticação por X-API-Key e registro de logs, construindo uma base sólida para a fase de coleta e armazenamento mencionada por ZACEPINS et al.

(2020). Ao mesmo tempo, o dashboard, a exportação em CSV e a responsividade da interface aproximam o sistema da fase de análise, ao fornecer visualizações históricas e mecanismos de extração de dados adequados à rotina de pesquisadores.

O trabalho de CARDOSO et al. (2023) é particularmente próximo deste projeto, ao propor uma solução IoT com transmissão via LoRa, gateway ESP32 e um front-end construído em Power Apps integrado ao Excel. Apesar dessa contribuição, os próprios autores relatam limitações na integração protótipo–software: dados de temperatura e umidade sendo recebidos como string (resultando em valores “nan” nos gráficos), leituras de peso fora de escala pela ausência de calibração e parte da interface descrita apenas como projeto ideal, não como implementação plenamente funcional. Em contraste, os resultados do Sistema Melissa mostram que a API de ingestão e a interface *web* operam sobre dados devidamente tipados e calibrados, permitindo gráficos históricos consistentes e geração de arquivos CSV prontos para análise em ferramentas externas. Enquanto CARDOSO et al. enfatizam as dificuldades do Power Apps em interpretar os dados de sensores, o Melissa resolve justamente essa lacuna ao entregar uma camada de apresentação estável, desenvolvida especificamente para o sistema e integrada de forma nativa ao banco de dados relacional.

Além disso, a solução proposta neste trabalho incorpora de forma explícita um componente de segurança física da colmeia, por meio da regra de negócio que gera alertas em caso de abertura não autorizada. Essa dimensão de segurança ativa, validada tanto na API (detecção do campo security.opened = true) quanto na interface (“Alertas Recentes” exibindo eventos críticos), não é descrita como foco central nos trabalhos de SILVA (2017), ZACEPINS et al. (2020), CARDOSO et al. (2023) ou MARQUES et al. (2024), que se concentram majoritariamente no monitoramento ambiental e na visualização de dados. Assim, os resultados indicam que o Sistema Melissa amplia o escopo típico das soluções de monitoramento apícola ao integrar monitoramento ambiental e segurança física em uma única plataforma *web*.

Por fim, ao dialogar com a agenda de desafios proposta por ZACEPINS et al. (2020), observa-se que o Melissa representa um grande passo na direção de sistemas de suporte à decisão em Apicultura de Precisão. Embora este trabalho não implemente ainda um módulo completo de recomendação ou automação de ações, a infraestrutura construída composta por *API* de ingestão validada, backend com regras de negócio, interface *web* responsiva e mecanismo de exportação de dados, atende às fases iniciais do ciclo (coleta e apresentação) de maneira mais completa do que parte das soluções correlatas. Isso cria um alicerce tecnológico sobre o qual futuras pesquisas podem desenvolver algoritmos analíticos, modelos preditivos e funcionalidades de tomada de decisão automatizada.

Em síntese, a discussão dos resultados mostra que o Sistema Melissa não apenas cumpre os requisitos definidos, mas também preenche lacunas identificadas na literatura, ao combinar robustez na ingestão de dados, interface *web* orientada ao uso científico e mecanismos de segurança física da colmeia. Esses diferenciais posicionam o artefato como uma contribuição relevante para o ecossistema de soluções IoT voltadas à apicultura de precisão.

6. CONSIDERAÇÕES FINAIS

O trabalho desenvolvido alcançou seu objetivo geral, que foi "criar uma interface *web* para um protótipo de colmeia inteligente que lide com os dados dos sensores em tempo real, desde o recebimento até o armazenamento e a exibição". O artefato de *software* desenvolvido demonstrou ser uma solução funcional e robusta. Os testes de *API* comprovaram que o sistema é capaz de receber dados (RF03), processar regras de negócio (RB03), persistir leituras (RF04) e registrar alertas (RF05) de forma eficiente, com latência de resposta (330 ms) bem abaixo do limite de 500 ms estipulado (RNF02).

Paralelamente, todos os objetivos específicos foram atingidos. O trabalho: a) Levantou os requisitos funcionais, não funcionais e regras de negócio, consolidados no Documento de Requisitos (Apêndice A). b) Projetou e implementou o banco de dados, conforme o modelo relacional apresentado. c) Desenvolveu a interface *web* responsiva, validando sua adaptabilidade em dispositivos móveis e a correta implementação dos *dashboards* e gráficos. d) Integrou o protótipo físico com a aplicação *web*, através de uma *API* de ingestão (*api/ingest.php*) segura, que valida os dispositivos por *X-API-Key* (RB01, RB02) e está pronta para receber os dados do *hardware*.

A conclusão deste projeto não encerra as possibilidades do Sistema Melissa. Pelo contrário, o artefato construído serve como uma plataforma-base para diversas expansões. A seguir, são apresentadas as propostas de trabalhos futuros.

Implementação de Notificações Ativas: atualmente, o sistema exige que o usuário consulte o *dashboard* (RF08) para verificar alertas (RF05). Uma evolução natural é a implementação de um serviço de notificações (via e-mail, SMS ou push) que alerte o pesquisador ativamente e em tempo real sobre eventos críticos, como uma abertura não autorizada (RB03).

Análise de Dados e Machine Learning: com o acúmulo de dados históricos na tabela readings (RF04), abre-se a possibilidade para análises preditivas. Trabalhos futuros poderiam aplicar técnicas de machine learning

para identificar padrões complexos, prever a saúde da colônia, ou detectar anomalias (como a perda de peso súbita) que poderiam passar despercebidas em gráficos simples.

Suporte a Múltiplos Apiários: embora o banco de dados tenha sido modelado para suportar múltiplas colmeias (Seção 4.2), a interface atual (Seção 5.1.2) foca na visualização de uma colmeia por vez. Um trabalho futuro poderia criar um *dashboard* "gerencial" que exiba um mapa ou uma visão consolidada do status de todas as colmeias de um ou mais apiários.

Evolução da Conectividade: a arquitetura atual depende de conectividade Wi-Fi (Seção 4.3.2), o que pode ser um limitador para a implantação em zonas rurais. Uma pesquisa futura de grande valor seria a adaptação do *hardware* e da arquitetura para utilizar redes de baixa potência e longo alcance (LPWAN), como LoRaWAN, ou redes celulares (4G/5G), aumentando drasticamente a viabilidade da aplicação em campo.

O desenvolvimento do Sistema Melissa, portanto, cumpre seu papel como um artefato de *software* completo, unindo os conceitos de Apicultura de Precisão e Internet das Coisas (Capítulo 2) em uma ferramenta acessível e de baixo custo (Seção 3.1). O trabalho contribui para o ecossistema do Projeto Melissa no Instituto Federal Baiano, Campus Catu, entregando uma plataforma validada, documentada (Capítulo 3) e escalável, pronta para as próximas fases de pesquisa e aplicação prática. O código-fonte do artefato, visando a proteção da propriedade intelectual associada ao projeto de pesquisa, encontra-se mantido em um repositório de versionamento privado, tendo seu acesso restrito aos membros do grupo de pesquisa da instituição.

Referências

ATZORI, L.; IERA, A.; MORABITO, G. *The Internet of Things: a survey*. *Computer Networks*, v. 54, n. 15, p. 2787-2805, 2010. DOI: 10.1016/j.comnet.2010.05.010. Disponível em: https://doi.org/10.1016/j.comnet.2010.05.010. Acesso em: 6 nov. 2025.

CARDOSO, D. F.; SILVA, F. S.; MAIA, M. V.; ANGELO, N. *Desenvolvimento de um protótipo de caixa de abelhas com IoT. Trabalho de Conclusão de Curso (Engenharia de Produção).* Instituto Mauá de Tecnologia, 2023. Disponível em: https://repositorio.maua.br/handle/MAUA/523. Acesso em: 12 nov. 2025.

CODD, E. F. *A relational model of data for large shared data banks*. *Communications of the ACM*, v. 13, n. 6, p. 377–387, 1970. DOI: 10.1145/362384.362685. Disponível em: https://doi.org/10.1145/362384.362685. Acesso em: 6 nov. 2025.

COTA, D.; MARTINS, J.; MAMEDE, H.; BRANCO, F. *BHiveSense: An integrated information system architecture for sustainable remote monitoring and management of apiaries based on IoT and microservices*. *Journal of Open Innovation: Technology, Market, and Complexity*, v. 9, n. 3, p. 100110, 2023. Disponível em: https://www.sciencedirect.com/science/article/pii/S2199853123002123. Acesso em: 6 nov. 2025.

DATE, C. J. *An Introduction to Database Systems*. 6. ed., ilust. Vol. 1 de Addison-Wesley Systems Programming Series. Boston: Addison-Wesley Publishing Company, 1995. 839 p. ISBN 020154329X; 978-0201543292. Disponível em: https://books.google.com.br/books?id=2xBRAAAAMAAJ#:~:text=to%20Database%20Systems-,C.%20J.%20Date,views%2C%20domains%20and%20missing%20information. Acesso em: 6 nov. 2025.

EMBRAPA Meio-Norte; PEREIRA, F. de M.; SAGRILO, E.; ALCÂNTARA, R. M. C. M. de (eds. téc.). *Anais da VI Jornada Científica da Embrapa Meio-Norte: VI Jornada Científica da Embrapa Meio-Norte, Teresina, PI, 25 a 27 de novembro de 2020.* Teresina: Embrapa Meio-Norte, 2021. 71 p. (Documentos / Embrapa Meio-Norte; ISSN 0104-866X; 284). Disponível em: https://www.infoteca.cnptia.embrapa.br/infoteca/bitstream/doc/1143459/1/VIJornadaCientificaEmbrapaMeioNorteDoc284.2021.pdf. Acesso em: 17 out. 2025.

FONTENELE, T. A. *Um sistema baseado em IoT para detecção e notificação de dano ou furto de colmeias de abelhas.* 2022. 107 f. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) – Universidade Federal do Ceará, Campus de Quixadá, Quixadá, 2022. Disponível em: http://repositorio.ufc.br/handle/riufc/65432. Acesso em: 6 nov. 2025.

GERONET SERVICES. *Apicultura: criação de abelhas e produção de mel.* Porto Alegre: Biblioteca AGPTEA, 2004. 119 p. Disponível em: https://www.bibliotecaagptea.org.br/zootecnia/apicultura/livros/APICULTURA%20CRIACAO%20DE%20ABELHAS%20E%20PRODUCAO%20DE%20MEL.pdf. Acesso em: 17 out. 2025.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. *Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems*, v. 29, n. 7, p. 1645-1660, 2013. DOI: 10.1016/j.future.2013.01.010. Disponível em: https://doi.org/10.1016/j.future.2013.01.010. Acesso em: 6 nov. 2025.

HEVNER, A. R.; MARCH, S. T.; PARK, J.; RAM, S. *Design science in information systems research. MIS Quarterly*, v. 28, n. 1, p. 75–105, 2004. Disponível em:

https://www.researchgate.net/publication/201168946_Design_Science_in_Information_Systems_Research. Acesso em: 7 nov. 2025.

KULYUKIN, V.; MUKHERJEE, S.; AMLATHE, P. *Toward audio beehive monitoring: deep learning vs. standard machine learning in classifying beehive audio samples. Applied Sciences*, v. 8, n. 9, p. 1573, 2018. DOI: 10.3390/app8091573. Disponível em: https://doi.org/10.3390/app8091573. Acesso em: 6 nov. 2025.

MACHADO, P. A. *Utilização de tecnologia de precisão na apicultura: uma revisão sistemática.* Trabalho de Conclusão de Curso (Bacharelado em Zootecnia). Faculdade de Agronomia, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2024. Disponível em: https://lume.ufrgs.br/handle/10183/276197. Acesso em: 12 nov. 2025.

MACIEL, M. A. de S.; CAETANO, A. L. L.; GUIMARÃES, P. V. de S.; JUCÁ, S. C. S. *Desenvolvimento de Sistema de Monitoramento IoT Utilizando Princípios de Aprendizagem Baseada em Projetos.* In: *WORKSHOP DE INFORMÁTICA NA ESCOLA (WIE)*, 23., 2017, Recife. *Anais* […]. Porto Alegre: Sociedade Brasileira de Computação, 2017. p. 462–469. DOI: 10.5753/cbie.wie.2017.462.

MARQUES, R.; FRANCO, W.; PINHEIRO, S. L.; MIRANDA, J. I. S.; FREITAS, E. D. G. de; SOUZA, R. W. R. de. *Sistema IoT de monitoramento de colmeias de abelhas Apis mellifera.* In: ESCOLA REGIONAL DE COMPUTAÇÃO DO CEARÁ, MARANHÃO E PIAUÍ (ERCEMAPI), 12., 2024, Parnaíba/PI. Anais [...]. Porto Alegre: *Sociedade Brasileira de Computação*, 2024. p. 297–302. DOI: 10.5753/ercemapi.2024.243787. Disponível em: https://doi.org/10.5753/ercemapi.2024.243787. Acesso em: 12 nov. 2025.

MELO, R. M. *Sistema de monitoramento de abelhas Apis mellifera.* Trabalho de Conclusão de Curso (Graduação em Ciência da Computação). Campus de Crateús, Universidade Federal do Ceará, Crateús, 2025. Disponível em: https://repositorio.ufc.br/handle/riufc/80027. Acesso em: 12 nov. 2025.

NOCELLI, R. C. F.; ROAT, T. C.; ZACARIN, E. C. M. da S.; MALASPINA, O. *Riscos de pesticidas sobre as abelhas.* In: *TERCEIRA SEMANA DOS POLINIZADORES*, 3., 2010, Petrolina, PE. *Palestras e resumos.* Petrolina: Embrapa Semiárido, 2012. p. 197–212. (Embrapa Semiárido. Documentos, 249). Disponível em: https://ainfo.cnptia.embrapa.br/digital/bitstream/item/69299/1/Roberta.pdf. Acesso em: 5 nov. 2025.

PEFFERS, K.; ROTHENBERGER, M.; KUECHLER, B. *A design science research methodology for information systems research. Journal of Management Information Systems*, v. 24, n. 3, p. 45–77, 2007. Disponível em: https://www.researchgate.net/publication/284503626_A_design_science_research_methodology_for_information_systems_research. Acesso em: 7 nov. 2025.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software: uma abordagem profissional.* 8. ed. Porto Alegre: AMGH, 2016. Disponível em:

https://archive.org/details/pressman-engenharia-de-software-uma-abordagem-profissional-8a. Acesso em: 7 nov. 2025.

RICKETTS, T. H. *et al. Landscape effects on crop pollination services: are there general patterns? Ecology Letters*, v. 11, p. 499-515, 2008. DOI: 10.1111/j.1461-0248.2008.01157.x. Disponível em: https://doi.org/10.1111/j.1461-0248.2008.01157.x. Acesso em: 6 nov. 2025.

SANTOS, J. O. *Um estudo sobre a evolução histórica da apicultura.* 2015. 95 f. Dissertação (Mestrado Profissional em Sistemas Agroindustriais) – Centro de Ciências e Tecnologia Agroalimentar, Universidade Federal de Campina Grande, Pombal, PB, 2015. Disponível em: https://dspace.sti.ufcg.edu.br/handle/riufcg/873. Acesso em: 6 nov. 2025.

SILVA, M. G.; GOMES, I. A.; MEDEIROS, A. C. de; SILVA, R. A. da; MARACAJÁ, P. B. *Apicultura brasileira: aspectos técnicos e práticos no manejo de abelhas africanizadas.* In: *Ciências Agrárias: Inovação, Tecnologia, Desenvolvimento e Extensão.* [S.l.]: GEPRA Editora, 2021. p. 190. Disponível em: https://www.researchgate.net/publication/354342381_APICULTURA_BRASILEIRA_ASPECTOS_TECNICOS_E_PRATICOS_NO_MANEJO_DE_ABELHAS_AFRICANIZADAS. Acesso em: 17 out. 2025.

SOMMERVILLE, I. *Engenharia de Software.* 10. ed. São Paulo: Pearson, 2019. Disponível em: https://www.facom.ufu.br/~william/Disciplinas%202018-2/BSI-GSI030-EngenhariaSoftware/Livro/engenhariaSoftwareSommerville.pdf. Acesso em: 7 nov. 2025.

ULLMAN, L. *PHP and MySQL for Dynamic Web Sites: Visual QuickPro Guide.* 5. ed. San Francisco: Peachpit Press, 2017. Disponível em: https://www.amazon.com.br/PHP-MySQL-Dynamic-Web-Sites/dp/0134301846. Acesso em: 7 nov. 2025.

VIANA, B. F.; SILVA, F. O. *Polinização por abelhas em agroecossistemas.* [S.l.: s.n.], 2010. Disponível em: http://www.apis.sebrae.com.br/Arquivos/16%C2%BA20Cong_Bras_Apic/Anais_1/POLINIZA%C3%87%C3%83O%20POR%20ABELHAS%20EM%20AGROECOSSISTEMAS.pdf. Acesso em: 6 nov. 2025.

ZANELLA, A.; BUI, N.; CASTELLANI, A.; VANGELISTA, L.; ZORZI, M. *Internet of Things for Smart Cities. IEEE Internet of Things Journal*, v. 1, n. 1, p. 22–32, 2014. DOI: 10.1109/JIOT.2014.2306328. Disponível em: https://doi.org/10.1109/JIOT.2014.2306328. Acesso em: 6 nov. 2025.

ZACEPINS, A.; BRUSBARDIS, V.; MEITALOVS, J.; STALIDZANS, E. *Challenges in the development of Precision Beekeeping. Biosystems Engineering*, v. 130, p. 60–71, 2015. DOI: 10.1016/j.biosystemseng.2014.12.001. Disponível em: https://doi.org/10.1016/j.biosystemseng.2014.12.001. Acesso em: 6 nov. 2025.

APÊNDICE A – DOCUMENTO DE REQUISITOS DO PROTÓTIPO DE INTERFACE WEB PARA COLMEIA INTELIGENTE

1 IDENTIFICAÇÃO

Projeto: Desenvolvimento de um protótipo de interface web para monitoramento de variáveis ambientais e segurança em colmeia inteligente.

Discente: Arthur Saldanha Félix Ulisses

Orientador: Gilvan Martins Durães

2 ESCOPO

O sistema consiste em um protótipo web que recebe dados de sensores instalados em uma colmeia experimental, transmitidos via *ESP8266*. O sistema deve:

Coletar e armazenar variáveis ambientais (temperatura, umidade, luminosidade, UV, peso).

Registrar eventos de segurança (abertura não autorizada da colmeia).

Disponibilizar informações por meio de uma interface web responsiva, com dashboard, gráficos e alertas.

Garantir autenticação tanto do dispositivo quanto dos usuários.

3 ATORES

Dispositivo (ESP8266): Envia leituras periódicas e eventos via HTTP POST.

Pesquisador: Visualiza leituras e alertas em *dashboard*.

Administrador: Gerencia colmeias, dispositivos e usuários.

4 REQUISITOS FUNCIONAIS (RF)

RF01 – Cadastrar colmeias com código, descrição e localização.

RF02 – Cadastrar dispositivos vinculados à colmeias, gerando API key única.

RF03 – Receber dados do *ESP8266* via *HTTP POST*, validando pela *API key*.

RF04 – Persistir leituras em banco, incluindo valores ambientais e *JSON* bruto.

RF05 – Registrar alerta *UNAUTHORIZED_OPEN* quando detectada abertura não autorizada.

RF06 – Atualizar automaticamente o campo last_seen do dispositivo a cada leitura recebida.

RF07 – Autenticar usuários por e-mail e senha, protegendo rotas internas.

RF08 – Exibir *dashboard* com últimas leituras, gráficos de variáveis ambientais e lista de alertas.

RF09 – Filtrar e exportar dados e alertas em CSV.

RF10 – Registrar logs de ingestão rejeitada (*payload*, status, motivo).

5 REQUISITOS NÃO FUNCIONAIS (RNF)

RNF01 – Todas as comunicações devem usar HTTPS.

RNF02 – Tempo de resposta da ingestão deve ser < 500 ms (p95).

RNF03 – Senhas de usuários devem ser armazenadas com hash seguro (bcrypt/argon2).

RNF04 – Banco relacional (MySQL) com integridade referencial.

RNF05 – Interface web deve ser responsiva (desktop e mobile).

RNF06 – *Endpoint* de ingestão deve aplicar rate limit para evitar abuso.

RNF07 – Logs devem registrar falhas de *API* e tentativas inválidas.

6 REGRAS DE NEGÓCIO (RB)

RB01 – Cada dispositivo possui uma API key única e obrigatória.

RB02 – Requisições sem X-API-Key válida devem ser rejeitadas (401).

RB03 – Se *security.opened* = *true*, o sistema gera automaticamente um alerta *CRITICAL*.

RB04 – Apenas dispositivos ativos (active=true) podem enviar leituras.

RB05 – Apenas usuários autenticados podem acessar *dashboards* e relatórios.

7 CONTRATO DE API – INGESTÃO

7.1 ENDPOINT

POST /api/ingest.php

7.2 HEADERS

Content-Type: application/json

X-API-Key: <chave_unica_do_dispositivo>

7.3 BODY (EXEMPLO)

JSON

```
{
  "hora": "27/09/2025 17:30:00",
  "temp_ext": 32.8,
  "umid_ext": 57.3,
  "lum_ext": 980,
  "temp_int": 30.2,
  "umid_int": 60.1,
  "lum_int": 850,
  "peso": 21.412,
  "calibracao": 0,
  "uv_index": 2.1,
  "security": { "opened": false }
}
```

7.4 RESPOSTAS

201 Created → leitura registrada.

401 Unauthorized → chave inválida/ausente.

403 Forbidden → dispositivo inativo.

422 Unprocessable Entity → *payload* inválido.

429 Too Many Requests → rate limit.

500 Server Error → falha interna.

8 MODELO DE DADOS (SIMPLIFICADO)

users (id, name, email, password_hash, role, created_at)

hives (id, code, description, location, created_at)

devices (id, hive_id, name, api_key, active, last_seen)

readings (id, device_id, ts, variáveis ambientais, raw_json)

alerts (id, device_id, hive_id, ts, type, severity, message)

ingest_logs (id, received_at, device_id, status_code, error_message, *payload*)

9 FLUXOS PRINCIPAIS

Provisionamento de dispositivo: admin cadastra colmeia → cadastra dispositivo → gera API key → grava API key no *ESP8266*.

Ingestão: ESP envia JSON com X-API-Key → API valida → grava leitura → se opened=true, gera alerta.

Visualização: usuário loga → acessa *dashboard* → consulta leituras/alertas → exporta dados.

10 CRITÉRIOS DE ACEITAÇÃO

Leitura com API key válida é registrada em readings com status 201.

Alerta de abertura é registrado em alerts automaticamente.

Leituras sem API key retornam 401 e são logadas.

Dashboard exibe últimas leituras, gráficos e alertas.

Usuário sem login não acessa rotas internas.

11 MÉTRICAS E AVALIAÇÃO

Taxa de ingestão bem-sucedida: > 99,5% (considerando uma margem para falhas de rede Wi-Fi).

Latência de ingestão: p95 < 500 ms (95% das requisições devem ser respondidas em menos de meio segundo).

Taxa de detecção de abertura: 100% (todos os eventos de abertura devem gerar um alerta *UNAUTHORIZED_OPEN*).