UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

JÚLIO CÉSAR MAIA DEOLINO

DESENVOLVENDO UM SISTEMA IOT USANDO DJANGO E UM BROKER MQTT PARA REALIZAR AUTOMAÇÃO NA AGRICULTURA

RUSSAS

2025

JÚLIO CÉSAR MAIA DEOLINO

DESENVOLVENDO UM SISTEMA IOT USANDO DJANGO E UM BROKER MQTT PARA REALIZAR AUTOMAÇÃO NA AGRICULTURA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Dr. Reuber Regis de Melo

RUSSAS

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

D465d Deolino, Júlio César Maia.
Desenvolvendo um sistema IoT usando Django e um broker MQTT para realizar automação na agricultura / Júlio César Maia Deolino. – 2025.
63 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Russas, Curso de Ciência da Computação, Russas, 2025.
Orientação: Prof. Dr. Reuber Regis de Melo.

1. internet das coisas. 2. automação. 3. MQTT. 4. Django. 5. agricultura inteligente. I. Título.
CDD 005

JÚLIO CÉSAR MAIA DEOLINO

DESENVOLVENDO UM SISTEMA IOT USANDO DJANGO E UM BROKER MQTT PARA REALIZAR AUTOMAÇÃO NA AGRICULTURA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em: 01/08/2025

BANCA EXAMINADORA

Prof. Dr. Reuber Regis de Melo (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Cenez Araújo de Rezende
Universidade Federal do Ceará (UFC)

Prof. Ms. Hugo Nathan Barbosa Regis
Universidade Federal do Ceará (UFC)

Dedico esse trabalho a Deus e aos meus pais, César Cals de Souza Deolino e Maria Lirete Maia da Costa.

AGRADECIMENTOS

Sou grato a Deus por me dar forças nesta caminhada e por todas as oportunidades que me concedeu.

Aos meus pais, César Cals de Sousa Deolino e Maria Lirete Maia, por todo o apoio e carinho que me foi dado.

Ao meu orientador, Prof. Dr. Reuber Regis de Melo, sem o qual não poderia ter realizado este trabalho.

Por fim, à Universidade Federal do Ceará (UFC), pela oportunidade de me aprimorar e obter um ensino de nível superior.

“ A maior glória em viver não reside em nunca cair, mas em levantar-se cada vez que caímos.”

(Nelson Mandela)

RESUMO

O avanço da tecnologia tem impactado diversas áreas, incluindo a agricultura, que precisa se modernizar para atender à crescente demanda mundial por alimentos. O aumento populacional impõe desafios como o controle de pragas, a utilização eficiente de recursos naturais (como água e fertilizantes) e o monitoramento climático. Diante desse cenário, a Internet das Coisas (Internet of Things (IoT)) surge como uma solução promissora para otimizar processos agrícolas. Este trabalho apresenta a continuidade do desenvolvimento do sistema *AgroInfo*, aprimorando o *backend* para possibilitar automação e integração de novas funcionalidades. O sistema atual dispõe de um sistema de cadastro de usuários, propriedades, plantio, e visualização de dados recebidos pelo sensor transmitidos através de um sistema *broker*. Com o desenvolvimento de novas funcionalidades é possível, além de receber dados, enviar comandos para atuadores como válvulas ou bombas o que torna possível realizar a automação de propriedades agrícolas, além de melhorar a escalabilidade, permitir o cadastro de sensores para cada plantio e aprimorar o gerenciamento de dados. O sistema utiliza plataformas embarcadas com sensores para coletar dados das plantações e atuadores que recebem comandos remotamente, que são disponibilizados via *broker* (MQTT), permitindo a tomada de decisões mais eficientes e práticas com base nos dados coletados. Para a implementação, é adotada a arquitetura MVT (*Model-Template-View*), que organiza a estrutura da aplicação, e utilizados diagramas *UML*, como casos de uso, sequência e classes, para representar a interação dos usuários e o fluxo do sistema. O desenvolvimento utiliza os *frameworks Django* e *Django Rest Framework*. Com isso, os testes preliminares vêm confirmando a viabilidade bidirecional via Message Queuing Telemetry Transport (MQTT) para a automação, afirmando a capacidade de leitura de dados e envio de comandos eficazes. Espera-se disponibilizar uma ferramenta que possa auxiliar na modernização da agricultura no Vale do Jaguaribe, proporcionando maior eficiência e sustentabilidade na gestão agrícola.

Palavras-chave: internet das coisas. Automação. MQTT. Django. Agricultura Inteligente.

ABSTRACT

The advancement of technology has impacted several areas, including agriculture, which needs to modernize to meet the growing worldwide demand for food. Population increase imposes challenges such as pest control, efficient use of natural resources (such as water and fertilizers) and climate monitoring. Given this scenario, the Internet of Things (IoT) emerges as a promising solution to optimize agricultural processes. This work proposes the continuity of the development of the *AgroInfo* System, improving *Backend* to enable automation and integration of new features. The current system has a user registration system, properties, planting, and viewing data received by the sensor transmitted through a *broker* system. This work will allow us to receive data, send commands to actuators such as valves or pumps which enables the automation of agricultural properties, as well as improving scalability, enabling the registration of sensor for each planting and improving data management. The system uses sensors embedded platforms to collect data from plantations and actuators who receive remote commands, which are made available via *Broker* MQTT, allowing more efficient and practical decisions based on the collected data. For implementation, Model, Template, View (MVT) (*Model-Template-View*) architecture, which organizes the application structure, and used *UML* diagrams, such as use cases, sequence and classes, to represent users interaction and system flow is adopted. Development uses *Django* and *Django Rest Framework frameworks*. As a result, preliminary tests have been confirming bidirectional viability via MQTT for automation, affirming the data reading capacity and sending effective commands, it is expected to make available a tool that can assist in the modernization of agriculture in the Jaguaribe Valley, providing greater efficiency and sustainability in agricultural management.

Keywords: internet of things. Automation. MQTT. Django. Intelligent Agriculture.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CP4E	Computer programming for everybody
INMET	Instituto Nacional de Meteorologia
IoT	Internet of Things
LCD	Liquid Crystal Display
MQTT	Message Queuing Telemetry Transport
MVT	Model, Template, View
NPK	Nitrogênio (N), Fósforo (P) e Potássio (K)
PHP	Hypertext Preprocessor
PIB	Produto Interno Bruto
RFID	Radio Frequency IDentification
RTC	Real-Time Clock
SCADA	Supervisory Control and Data Acquisition

SUMÁRIO

1 INTRODUÇÃO

De acordo com pesquisas realizadas pela Organização das Nações Unidas para a Alimentação e Agricultura (FAO), estima-se que até o ano de 2050 o crescimento populacional mundial alcance aproximadamente 9 bilhões de pessoas, o que exigirá um aumento de 60% na produção de alimentos para suprir a demanda global (FAO, 2018). Esse cenário impõe desafios significativos para o setor agrícola, que precisa modernizar-se para garantir maior eficiência na produção e uso sustentável dos recursos naturais.

Um dos principais desafios para a produção de alimentos em larga escala está na otimização do uso de insumos agrícolas, na redução de desperdícios e na melhoria da produtividade. A IoT (Internet das Coisas) surge como uma solução promissora para enfrentar esses desafios, pois permite a coleta e análise de dados ambientais em tempo real, além da automação de processos agrícolas. Essa abordagem viabiliza práticas mais eficientes, como o uso racional da água, fertilizantes e defensivos agrícolas, garantindo maior produtividade e sustentabilidade no campo (Godfray, 2010).

A transformação digital vem evoluindo rapidamente, tornando a tecnologia mais acessível à população. A internet, por exemplo, é uma das inovações que mais impactaram o cotidiano, possibilitando o desenvolvimento de novas soluções tecnológicas (Jesus, 2021). No entanto, muitos pequenos agricultores ainda enfrentam dificuldades para acessar essas inovações devido a barreiras financeiras e estruturais. No Brasil, 76,8% das propriedades rurais são caracterizadas como agricultura familiar (IBGE, 2017), e grande parte dessas propriedades carece de acesso a tecnologias acessíveis e eficazes para modernizar sua produção (Gomes, 2023).

Diante desse contexto, este trabalho propõe a continuidade do desenvolvimento do sistema *AgroInfo*, focado na implementação de novas funcionalidades no *backend* para permitir a automação de processos agrícolas e a integração com sensores para monitoramento ambiental. Para isso, serão analisados o protocolo MQTT (Message Queue Telemetry Transport), utilizado para a comunicação eficiente entre dispositivos IoT, e o *framework Django*, empregado no desenvolvimento da aplicação. Esse sistema será projetado para oferecer uma solução acessível aos agricultores do Vale do Jaguaribe (Região localizada no centro-leste do Ceará), permitindo que tenham acesso a dados precisos para otimizar suas decisões propiedades.

1.1 Motivação

A agricultura é um dos setores mais importantes da economia brasileira, representando até 21% do Produto Interno Bruto (PIB) nacional (CEPEA, 2023). Além disso, a agricultura familiar desempenha um papel fundamental na produção de alimentos no país, correspondendo a 60% da produção total de alimentos consumidos internamente (IBGE, 2017).

Apesar de sua relevância, muitos agricultores enfrentam dificuldades para adotar tecnologias modernas devido aos altos custos e à falta de infraestrutura digital. Pequenos produtores, em especial, possuem recursos limitados para investir em soluções tecnológicas, o que impacta diretamente sua produtividade e eficiência operacional. Para esses agricultores, a adoção de ferramentas tecnológicas acessíveis pode ser determinante para a otimização da produção e para a adoção de práticas mais sustentáveis (Gomes, 2023).

Dessa forma, o presente trabalho busca desenvolver uma solução de baixo custo para agricultores da região do Vale do Jaguaribe, utilizando IoT para automatizar processos agrícolas e fornecer informações estratégicas que possibilitem a melhoria na gestão das lavouras.

1.2 Objetivo

1.2.1 Objetivo geral

Este trabalho tem como objetivo adicionar funcionalidades ao *backend* de um sistema IoT de gestão agrícola, permitindo a automação de processos no campo e a integração com sensores para coleta de dados ambientais via *broker* MQTT.

1.2.2 Objetivos específicos

- Realizar uma revisão bibliográfica sobre soluções tecnológicas aplicadas à agricultura digital;
- Estudar e analisar o *framework Django* e suas funcionalidades para o desenvolvimento do *backend*;
- Revisar o funcionamento do protocolo MQTT e sua arquitetura *Publish/Subscribe* para comunicação entre dispositivos IoT;
- Projetar e implementar novas funcionalidades no sistema, permitindo a automação agrícola baseada na comunicação MQTT.

1.3 Organização do trabalho

Este trabalho está estruturado em seis capítulos, além da introdução e conclusão.

No **Capítulo 2 – Fundamentação Teórica**, são descritas as principais tecnologias utilizadas no desenvolvimento da aplicação, incluindo conceitos sobre IoT, MQTT, *Django* e agricultura digital.

No **Capítulo 3 – Trabalhos Relacionados**, são apresentados estudos e projetos semelhantes, destacando suas abordagens e contribuições para o setor agrícola.

No **Capítulo 4 – Metodologia**, são detalhados os passos seguidos para o desenvolvimento do sistema, incluindo a estruturação da aplicação, modelagem dos dados e integração com sensores.

No **Capítulo 5 – Resultados**, são detalhados os passos dos testes realizados na aplicação, além de detalhar as ferramentas utilizadas.

No **Capítulo 6 – Conclusões Trabalho Futuros**, Apresentaremos a conclusão final e ideias para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Agricultura no Brasil

O Brasil é responsável por boa parte da produção de alimentos no mundo. No ano de 2021, o agronegócio foi responsável por 27,4% do PIB brasileiro e representou 48% do total das exportações do país (Caligaris, 2022).

Nos últimos anos, a modernização no setor agrícola tem se intensificado com o crescimento no uso de tecnologias como sensores, drones e inteligência artificial. Essas inovações são essenciais para melhorar a eficiência das plantações, aumentar a produtividade e diminuir os gastos de água e terras.

Com o avanço da IoT e dos dispositivos conectados, as fazendas inteligentes vêm se tornando realidade, o que permite aos agricultores tomarem decisões mais precisas, pois contam com o apoio das variáveis obtidas pelos sensores como umidade, luminosidade, temperatura, entre outros, além de poder automatizar algumas atividades que antes eram manuais.

Apesar dos avanços tecnológicos, o Brasil ainda enfrenta uma infraestrutura precária e acesso limitado à internet, o que gera uma série de desafios a serem superados, juntamente com a falta de capacitação para a utilização de novas tecnologias.

2.2 Internet das Coisas (IoT)

O termo IoT (*Internet of Things*) é usado para se referir a infraestruturas tecnológicas e serviços. No tocante à infraestrutura, a IoT pode ser definida como uma rede global de dispositivos inteligentes, que são conectados à internet e dependentes de tecnologias de processamento, comunicação e sensores como a Radio Frequency IDentification (RFID) (*Radio Frequency IDentification*) e as Redes de Sensores Sem Fio (*Wireless Sensor Networks*). Eles utilizam essas tecnologias para se comunicarem de maneira independente entre dispositivos e usuários finais (Silva, 2020).

As tecnologias mais importantes para a IoT são a tecnologia de identificação por radiofrequência (RFID) e a de sensores sem fio (*Wireless Sensor Network*). A RFID é utilizada para permitir que microchips transmitam informações de identificação para leitores sem fio de outros dispositivos que compartilhem da mesma tecnologia. Os sensores são utilizados de forma interconectada para monitorar e detectar as variáveis do ambiente (Silva, 2020).

Figura 1 – Conceito da Internet das Coisas

Fonte: (CODEMEC, 2016).

2.3 Computação em Nuvem

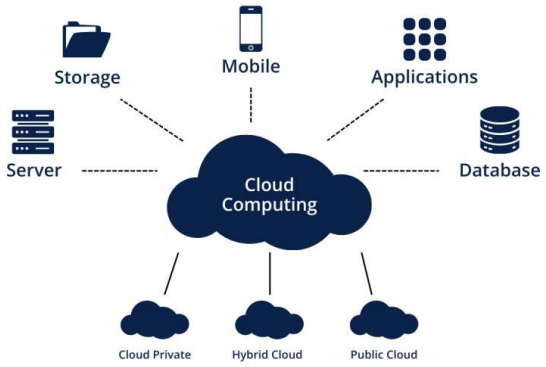
A computação em nuvem pode ser definida como a utilização massiva da virtualização para fornecer recursos computacionais sob demanda, por meio da internet. As tecnologias já existentes usadas no dia a dia foram adaptadas para realizar um novo modelo de acesso aos recursos computacionais (Carissimi; Barbosa, 2016).

No início, o poder de processamento em nuvem passou por ciclos de centralização e descentralização. Na década de 60, o poder computacional de processamento e armazenamento era concentrado em um único grande *mainframe*, e o acesso aos recursos era feito por terminais. Após isso, com o desenvolvimento dos computadores pessoais, houve a descentralização do poder da computação em nuvem, permitindo que os usuários processassem dados e armazenassem em seus próprios computadores (Carissimi; Barbosa, 2016).

Com a infraestrutura fornecida pela computação em nuvem, as plataformas de softwares como serviço são beneficiadas pela virtualização, pois ela permite a execução de várias instâncias em um mesmo *hardware*. Com os custos dos *data centers* são reduzidos, o que favorece a computação verde.

Um dos conceitos-chave da computação em nuvem é a elasticidade, que permite alocar recursos conforme a demanda. Um exemplo disso é o comércio eletrônico, onde são alugados os recursos necessários e a empresa não precisa investir em recursos próprios. “Pague o quanto usa” (*pay-as-you-go*) é o modelo responsável por flexibilizar o uso da computação em nuvem, permitindo que o cliente pague somente pelos recursos desejados.

Figura 2 - Arquitetura da Computação em Nuvem



Fonte: (Planton, 2024).

2.4 Python

No ano de 1982, em Amsterdã, capital da Holanda, Guido Van Rossum, que trabalhava no desenvolvimento da linguagem de programação ABC, estava envolvido no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação em um sistema chamado Amoeba. Devido a grandes falhas no sistema, que era feito na linguagem C, Guido resolveu desenvolver sua própria linguagem. Ele queria criar uma tecnologia fácil e intuitiva, pois alguns sistemas programados em C, por terem uma codificação extensa, apenas programadores experientes conseguiam entender.

Na década de 90, o projeto Python foi concluído. Guido então se mudou para os EUA, onde iniciou o projeto Computer programming for everybody (CP4E) (*Computer Programming for Everybody*), no qual ele ensinava programação de forma acessível. No ano de 2001, foi fundada a *PSF* (*Python Software Foundation*), que é uma organização sem fins lucrativos que coordena o uso da linguagem.

Com o tempo, o Python teve evoluções em sua estrutura e foram adicionados componentes como as compreensões de lista, funções e diversos outros. Hoje é altamente usada no mercado e já vem instalada em alguns sistemas operacionais como Linux e macOS.

2.5 Django Rest Framework

O *framework Django REST Framework* dispõe de uma biblioteca que estende a capacidade do Django, tornando-o mais eficiente e ágil para criar Application Programming Interface (API)s para a internet. Ela dispõe de ferramentas que são versáteis e modulares (Dantas, 2023).

Este *framework* é de código aberto, criado em Python, e tem seu foco em agilizar o desenvolvimento, diminuindo as duplicações no código, disponibilizando assim mais tempo para que os desenvolvedores se concentrem em tarefas mais cruciais (Dantas, 2023).

É amplamente utilizado por conta de suas vantagens, uma vez que oferece estruturas prontas para identificação de usuário e administração de conteúdo. Usa o padrão MVT e a linguagem Python, o que permite a criação de aplicações com poucas linhas de código (Carmo, 2023).

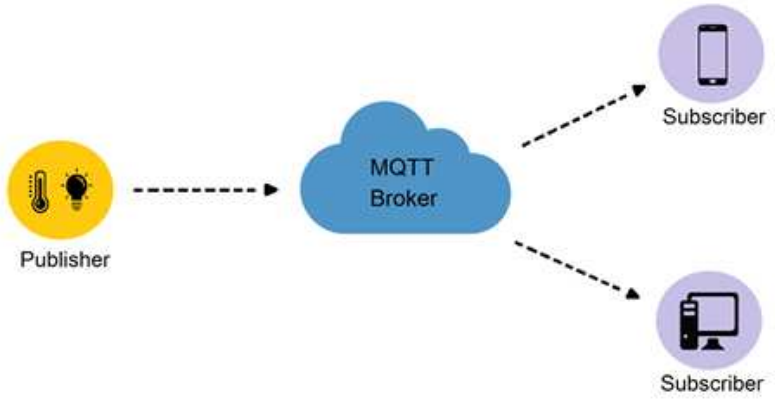
2.6 Protocolo MQTT

MQTT é um protocolo responsável por transmitir a comunicação principalmente entre dispositivos IoT. Foi projetado para ter uma simples implementação e poder transmitir os dados mesmo com uma rede de baixa qualidade. Este protocolo segue o modelo *Publish/Subscribe*, que é amplamente utilizado para dispositivos que têm limitações, principalmente com relação a armazenamento, processamento de dados e largura de banda.

O protocolo MQTT segue o modelo *Publish/Subscribe*, onde os dispositivos não se comunicam diretamente, mas há um intermediário entre eles: o *broker*. Os publicadores (*publishers*) são os que enviam as mensagens para o *broker*, que as organiza de forma hierárquica. Os assinantes (*subscribers*) se inscrevem no tópico de interesse e recebem as mensagens correspondentes.

Este protocolo é amplamente usado por dar suporte a curtas mensagens. Ele também pode ser utilizado em redes com alta latência e não confiáveis, já que o publicador e o assinante não necessitam estar online ao mesmo tempo. Ele se adequa a dispositivos limitados tanto em armazenamento, processamento de dados, quanto em limitações de largura de banda.

Figura 3 - Fluxo de Mensagens MQTT



Fonte: (Lena; Oliveira, 2018).

2.7 Arquitetura *Publish-Subscribe*

Nesta arquitetura, as mensagens são chamadas de eventos. Os componentes são os publicadores (*publishers*) e os assinantes (*subscribers*) de eventos. Os publicadores enviam mensagens que são executadas no *broker*. Os assinantes devem assinar previamente os eventos de seu interesse.

Em alguns sistemas que utilizam a arquitetura *Publish/Subscribe*, os eventos podem ser organizados com base em tópicos. Quando o publicador envia um evento, ele informa seu tópico, assim os clientes não precisam assinar todos os eventos do sistema.

3 TRABALHOS RELACIONADOS

Neste capítulo, serão abordados trabalhos com uma proposta semelhante ao tema escolhido, com o objetivo de comparar as tecnologias utilizadas em outras obras. Na seção 3.1 é apresentada uma solução tecnológica para a automação de pivôs e canais de irrigação. A seção 3.2 apresenta um sistema para monitoramento de variáveis climatológicas. A seção 3.3 apresenta um sistema IoT e Supervisory Control and Data Acquisition (SCADA) para um sistema de monitoramento de baixo custo. A seção 3.4 aborda um Sistema IoT baseado em *ESP32* para o controle e monitoramento de cultura em estufas. Por fim, a seção 3.5 apresenta uma análise sobre os trabalhos relacionados e este trabalho, comparando as tecnologias utilizadas e as escolhas de desenvolvimento.

3.1 IoT na Agricultura - Automação de Pivôs e Canais de Irrigação com Arduino e *Webservice*

Este artigo aborda os problemas e necessidades enfrentadas pelos produtores agrícolas, pois mesmo a agricultura sendo um setor produtivo e evoluído, também apresenta um déficit quanto à aplicação de tecnologia que possa promover a agricultura de precisão e a automação de processos produtivos. Dentre alguns dos motivos para não haver o aproveitamento em larga escala da tecnologia, podemos citar o alto custo e a possibilidade de vantagens alcançadas ainda serem prematuras. Artefatos computacionais integrados, como sensores de coleta e mecanismos de tomadas de decisão, têm se tornado mais acessíveis a partir dos últimos anos (Muxito, 2018).

Este trabalho tem como principal objetivo experimentar uma proposta de melhoria e controle de um sistema de irrigação na agricultura de precisão, de forma que possa ser de fácil implantação e de baixo custo. É proposto utilizar IoT para a construção de um protótipo que simula e testa os pivôs juntamente com uma nova solução tecnológica de monitoramento. O protótipo é composto por três ambientes: o ambiente mecânico, que é automatizado a partir da plataforma *Arduino*; o ambiente central, que se baseia no *Webservice* para integração; e também o ambiente de controle funcional, que se trata de um sistema *web* (Muxito, 2018).

O autor descreve as aplicações para a resolução do problema que foram um conjunto de elementos: foram escolhidos três módulos: o Real-Time Clock (RTC) (*RTC DS1307*), implementado para possibilitar a sincronização e a programação da data e hora para irrigação; o *ESP8266 ESP-01* de *Wi-Fi*, que permite a criação de *Webservice* e o acesso do sistema em longo

alcance (nele, os dados fornecidos pelo sensor e pelo RTC (*dateTime*) podem ser usados para *upload* e acessados por qualquer dispositivo conectado à internet); e o *I2C*, que funciona como adaptador para Liquid Crystal Display (LCD) e permite uma conexão mais fácil e econômica dos *jumpers* e dos pinos do *Arduino*. Além dos módulos, foram utilizados *Display* LCD como monitor, e botões físicos e operacionais no circuito, sendo cada botão responsável por uma determinada função (Muxito, 2018).

O autor descreve os principais *softwares* de apoio utilizados que foram: *Visual Studio Code v.1.26.1*, um editor de código-fonte gratuito; a linguagem Hypertext Preprocessor (PHP) *v.5.4* (PHP, 2015), linguagem de *scripts* de propósito geral que é especialmente adequada para o desenvolvimento *web*; e o banco de dados *MySQL v5.5* (MySQL, 2015), um sistema de gerenciador de banco de dados de código aberto (Muxito, 2018).

O autor conclui enfatizando a contribuição direta para a área da computação e a preocupação com os problemas e desafios na agricultura, destacando o envolvimento e colaboração a partir de parcerias realizadas junto às empresas de tecnologia que possibilitam o projeto.

3.2 Sistema IoT para monitoramento de variáveis climatológicas em culturas de agricultura urbana

Este artigo tem como objetivo o desenvolvimento de um sistema IoT para monitoramento de variáveis climáticas de interesse agrícola em áreas urbanas ou em hortas caseiras. Também pode servir como referência para implementação de sistemas de rastreamento e monitoramento IoT em cenários de agricultura urbana, de forma que possam ser customizados para características de diferentes culturas (Golondrino, 2022).

O autor propõe a utilização da arquitetura IoT de quatro camadas (captura, armazenamento, análise e visualização):

- **Camada de captura:** através de sensores, são adquiridas algumas variáveis como temperatura, umidade e luminosidade, que são enviadas para uma placa de captura que utiliza um pequeno servidor para leitura das informações;
- **Camada de armazenamento:** as variáveis são solicitadas à placa de captura e são armazenadas em um banco de dados relacional;
- **Camada de análise:** os dados são usados para medidas estatísticas e como modelos de aprendizagem;
- **Na camada de visualização:** é possível acompanhar as variáveis de interesse e as análises

estatísticas (Golondrino, 2022).

Para a metodologia de trabalho, o autor dividiu o trabalho em quatro fases metodológicas:

- **Seleção de ferramentas e tecnologias:** que se deu pela seleção de um conjunto de ferramentas e tecnologias e *softwares* livres para a captura de variáveis úteis na agricultura urbana;
- **Design da arquitetura IoT:** por meio de especificações baseadas nas quatro camadas convencionais dos sistemas que utilizam IoT (captura, armazenamento, análise e visualização);
- **Construção do protótipo do sistema:** com base nas análises das camadas e tecnologias definidas;
- **Estudo de caso:** aplicação prática do protótipo.

O autor conclui que o uso do sistema IoT foi útil para monitoramento de variáveis climatológicas em agricultura urbana. Embora tenha sido utilizado apenas em plantações de alface, os modelos podem ser usados em diferentes culturas de alimentos.

3.3 Utilização da IoT na agricultura sustentável

O objetivo desse artigo é desenvolver um sistema de monitoramento de baixo custo usando tecnologias como IoT e SCADA (*Supervisory Control and Data Acquisition*) para auxiliar pequenos agricultores que enfrentam dificuldades climáticas e financeiras, tendo em vista que o grande desafio da agricultura até os anos 2050 será o aumento da demanda de alimentos por conta do crescimento populacional, segundo a Organização das Nações Unidas para Alimentação e Agricultura (Gomes, 2023).

Para beneficiar de forma acessível e eficaz os pequenos agricultores familiares, o autor propõe a criação de um ambiente de monitoramento *open source* com base num sistema SCADA integrado a sensores. A finalidade desse sistema é melhorar a precisão das coletas de dados de umidade e temperatura do solo. Pequenos agricultores podem se beneficiar enormemente dessa solução que pode potencializar a eficácia e sustentabilidade de suas plantações (Gomes, 2023).

O sistema proposto utilizou um servidor *web Apache Tomcat*. Os dados são coletados por meio de sensores de vazão, umidade e temperatura, e foram armazenados em um banco de dados relacional *MySQL* que é integrado a um sistema SCADA. Para testar o programa, ele

utilizou o sistema em uma horta experimental no Campus de Passos. Os sensores foram usados para coletar os dados que foram comparados com os dados do Instituto Nacional de Meteorologia (INMET) (*Instituto Nacional de Meteorologia*) para garantir a precisão (Gomes, 2023).

O autor conclui que os resultados obtidos são confiáveis e podem ser usados para tomadas de decisões mais concisas e eficientes na agricultura. A utilização do sistema IoT é viável e eficaz, fazendo com que haja maior aproveitamento da água nas plantações, reduzindo o desperdício e o custo. Ele ainda planeja incluir sensores de *pH*, Nitrogênio (N), Fósforo (P) e Potássio (K) (NPK) e gases para auxiliar as propriedades sustentáveis.

3.4 Sistema IoT baseado em *ESP32* para o controle e monitoramento de cultura em estufas com foco na agricultura 4.0

O artigo tem como objetivo desenvolver um sistema *mobile* baseado no *ESP32* para realizar o monitoramento de variáveis como temperatura, umidade do ambiente, além do nível de água para irrigação em estufas de cultivo de alface. O sistema possibilita otimizar o gasto de água, aumentando assim a produtividade. Os fatores que podem limitar o crescimento da agricultura são principalmente a escassez de água e terra, sendo assim, no futuro, os cultivos dependerão da mecanização das atividades, inclusive das de pequena escala (Berrios, 2022).

O autor propõe a criação de um sistema IoT utilizando o microcontrolador *ESP32* para monitorar e acessar os dados gerados no cultivo de alface em estufas, com o objetivo de otimizar o uso dos recursos agrícolas. É utilizado o sistema supervisório *ScadaBR* e algumas tecnologias como *Firebase Real Time Database* e sensores que permitem uma gestão inteligente conectada (Berrios, 2022).

Na metodologia, realizou-se um estudo das características do cultivo em estufas e estabeleceram-se as restrições necessárias para as condições ideais para as plantações. Para realizar a medição das variáveis (temperatura, umidade, ambiente, umidade do solo e nível de líquido no tanque de água), foi utilizado o *ESP32-DeviKitC*, sensores e atuadores de baixo custo, e a tecnologia sem fio *ESP-NOW* para comunicação. A solução consiste em uma rede tipo estrela com nó central e três nós sensores. É utilizado *Firebase Realtime Database*, uma aplicação móvel baseada em *Android Studio*. Os nós sensores coletam os dados que são enviados para o nó central e depois recolhidos na base de dados, acionando ou desativando atuadores conforme o necessário (Berrios, 2022).

O autor conclui que o sistema IoT baseado no microcomputador *ESP32* e na rede

sem fio de sensores sob protocolos *ESP-NOW* cumpre seu papel de monitorar e controlar a temperatura, umidade e nível de água de irrigação, além de mostrar os dados em tempo real através de um aplicativo *Android*.

3.5 Comparação entre os trabalhos relacionados

Este trabalho trata-se da continuação do trabalho SANTOS (2024), onde foi selecionada a arquitetura MVT e o *framework* Django, que foi considerado a melhor opção para o desenvolvimento do *backend* do sistema *AgroInfo*. A aplicação do sistema foi realizada em Python, o que proporcionou um código mais reduzido e claro. O *framework* também oferece uma interface de controle e gerenciamento, com uma página de administração, sendo apenas preciso definir o banco de dados a ser utilizado, que no caso foi o *SQLite3*. A hospedagem foi mantida em nível local.

Tabela 1 – Comparativo dos trabalhos relacionados

Característica	**Este Trabalho**	**Ezequiel (2018)**	**Gabriel (2022)**	**Gomes (2023)**	**Berrios (2022)**
Arquitetura de Software	MVT	Camadas	Camadas	Client/Serv	Camadas
Framework / Plataforma	*Django*	*Webservice*	*Flask*	—	*Firebase*
Linguagem	*Python*	PHP	*Python*	*Java*	*C++*
Armazenamento	*SQLite3*	*MySQL*	*TinyDB*	*MySQL*	*Firebase*
Hospedagem	Local	Nuvem	Nuvem	Nuvem	Nuvem

Fonte: Elaborado pelo autor.

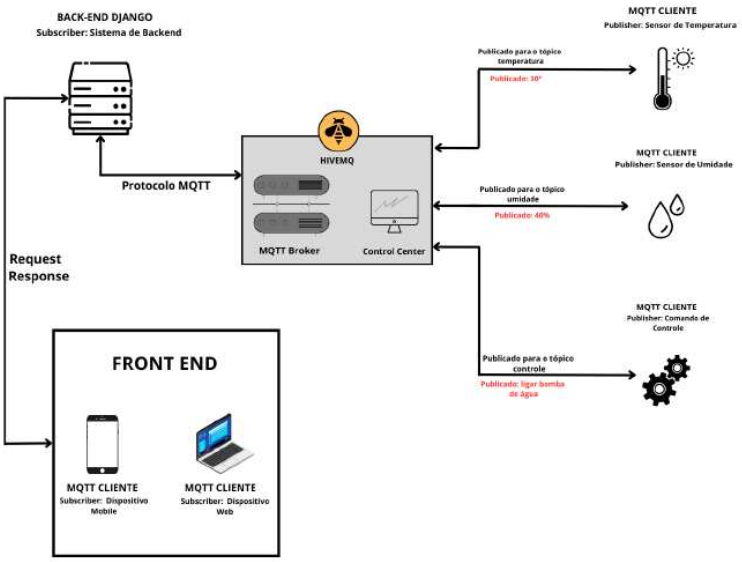
4 METODOLOGIA

4.1 Visão geral do trabalho

Este trabalho tem como objetivo dar continuidade ao trabalho desenvolvido pelo autor (Santos, 2024). Para a realização do trabalho, houve uma revisão bibliográfica e análise de trabalhos relacionados.

Na Figura 4 é ilustrada a representação da visão geral de todo o sistema *backend*, considerando as funcionalidades e as escolhas de arquitetura.

Figura 4 - Visão geral do sistema AgroInfo V1

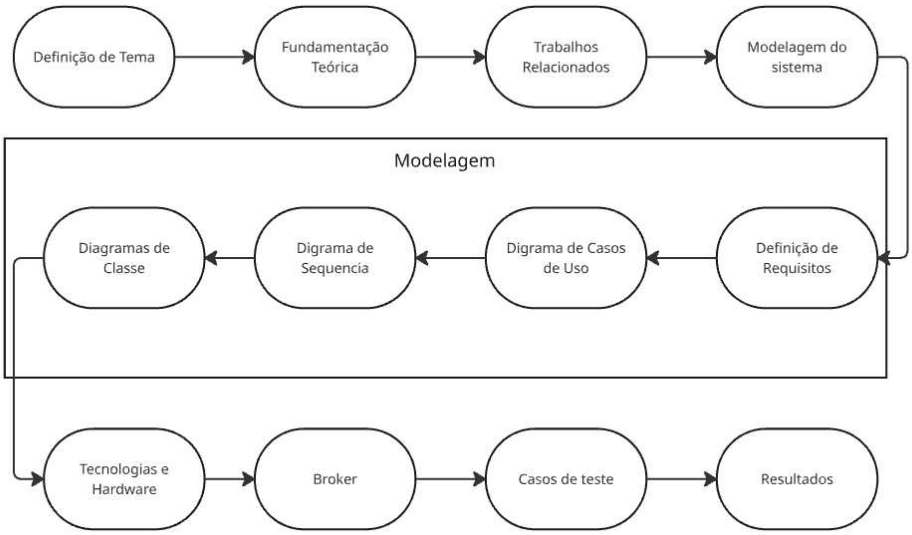


Fonte: (Santos, 2024).

A Figura 4 mostra o principal protocolo utilizado para comunicação entre os sensores, *broker* e o *backend*. Nesse caso, foi utilizado o protocolo MQTT, do tipo *Publish/Subscribe*, para possibilitar a troca de mensagens entre diversos dispositivos. E a arquitetura Cliente/Servidor foi usada para o processamento de dados. A arquitetura é capaz de se comunicar entre diversos dispositivos.

A Figura 5 mostra os passos seguidos para adicionar as funcionalidades definidas ao sistema com base na arquitetura definida. Serão especificados os passos para o desenvolvimento das funcionalidades adicionadas, juntamente com o resumo geral do sistema e o detalhamento dos testes realizados para a validação.

Figura 5 – Passos



Fonte: Elaborado pelo autor.

4.2 Levantamento de requisitos

Para definir os requisitos do sistema, foi realizada uma análise comparativa com estudos prévios na área, incluindo os trabalhos de (Muxito, 2018), (Golondrino, 2022), (Gomes, 2023) e (Berrios, 2022). A partir dessas análises, foram identificadas funcionalidades relevantes que poderiam ser incorporadas ao *AgroInfo*, tornando-o mais eficiente para auxiliar os agricultores do Vale do Jaguaribe.

Os requisitos do sistema foram classificados em funcionais e não funcionais, conforme descrito a seguir.

4.2.1 Requisitos Funcionais

Os requisitos funcionais representam as funcionalidades essenciais que o sistema deverá oferecer para atender às necessidades dos usuários. Foram definidos os seguintes requisitos:

4.2.1.1 Alerta de Falhas

O sistema deve ser capaz de identificar quando há uma falha na comunicação com o sensor e notificar no sistema.

4.2.1.2 Automação da Irrigação

O sistema deve permitir o acionamento e desligamento automático dos atuadores, além de permitir a configuração de horários programados.

4.2.1.3 Filtragem de Registros

O sistema deve possibilitar a filtragem dos registros armazenados por data e hora.

4.2.1.4 Registro de Irrigação

O sistema deve manter um histórico de acionamentos do sistema de irrigação.

4.2.1.5 Cadastrar Sensor

O sistema deve permitir cadastrar um sensor para cada plantio registrado no sistema, definindo por nome, tipo, *ID* e plantio correspondente.

4.2.1.6 Cadastrar Atuador

O sistema deve permitir cadastrar um atuador para cada plantio registrado no sistema, definindo por nome, tipo, *ID* e plantio correspondente.

4.2.2 Tabela de identificação e especificação de requisitos funcionais

A Tabela 2 apresenta a identificação e prioridade atribuída a cada requisito funcional, facilitando a rastreabilidade durante o desenvolvimento e os testes.

Tabela 2 - Tabela de identifição e especificação de requisitos funcionais

Identificação	**Nome da Funcionalidade**	**Prioridade**
RF001	Alerta de falhas	Essencial
RF002	Automação da Irrigação	Importante
RF003	Filtragem de registros	Importante
RF004	Registro de Irrigação	Importante
RF005	Cadastrar Sensor	Essencial
RF006	Cadastrar Atuador	Essencial

Fonte: (Elaborado pelo autor).

4.2.3 Requisitos não Funcionais

Os requisitos não funcionais referem-se a características desejáveis do sistema relacionadas à sua *performance*, confiabilidade e escalabilidade. Foram estabelecidos os seguintes requisitos:

- RNF001 – Desempenho da Leitura: O sistema deve processar e exibir as leituras ambientais em tempo real.
- RNF002 – Desempenho da Irrigação: O tempo de resposta para ativação e desligamento do sistema de irrigação não deve ultrapassar 2 segundos.
- RNF003 – Confiabilidade e Disponibilidade: O sistema deve operar de forma contínua, garantindo disponibilidade 24 horas por dia, 7 dias por semana.
- RNF004 – Recuperação: Em caso de falhas, o sistema deve ser capaz de reiniciar automaticamente e retomar sua operação.
- RNF005 – Escalabilidade: O sistema deve ser projetado para suportar um número crescente de sensores sem perda significativa de desempenho.
- RNF006 – Manutenibilidade: O código deve seguir boas práticas de desenvolvimento para facilitar futuras atualizações e correções.
- RNF007 – Suporte a Múltiplos Sensores/Atuadores: O sistema deve permitir a conexão simultânea de diversos sensores/atuadores sem comprometer a eficiência da comunicação.

4.2.4 Tabela de identificação e especificação de requisitos não funcionais

A Tabela 3 apresenta a especificação desses requisitos e suas respectivas prioridades no desenvolvimento do sistema.

Tabela 3 - Tabela de identifição e especificação de requisitos não funcionais

Identificação	**Nome da funcionalidade**	**Prioridade**
RNF001	Desempenho da leitura	Essencial
RNF002	Desempenho da Irrigação	Desejável
RNF003	Confiabilidade e Disponibilidade	Essencial
RNF004	Recuperação	Importante
RNF005	Escalabilidade	Importante
RNF006	Manutenibilidade	Desejável
RNF007	Múltiplos Sensores/Atuadores	Essencial

Fonte: (Elaborado pelo autor).

4.3 Relacionamento Geral do Sistema

Identificamos os principais requisitos e aspectos necessários para desenvolver o sistema de maneira que seja de fácil acesso e escalável. Na Figura **??** temos a visão geral do sistema e com base nas funcionalidades definidas.

Isso possibilita a escalabilidade tanto de processos quanto de sensores atuando ao mesmo tempo, e a possibilidade de executar tarefas em segundo plano sem consumir muitos recursos.

4.3.1 Diagrama de Fluxo Geral do Sistema

Este diagrama de componentes ilustra o fluxo de dados do sistema, com as novas relações do *Celery Broker*. Demonstra como os diferentes elementos do sistema interagem para coletar, processar e enviar comandos.

Para facilitar a visualização das relações, dividimos o diagrama em áreas funcionais:

- **Dispositivos IoT:**
  - **Sensores:** dispositivos que coletam dados do ambiente (temperatura, umidade, luminosidade).
  - **Atuadores:** dispositivos que recebem comandos do sistema e executam ações no ambiente.
- **Serviços de Fila:**
  - **MQTT *Broker* (*HiveMQ*):** Atua como "correio" central para mensagens IoT. Os sensores enviam os dados para ele, que os distribui para o *backend* e também é responsável por receber comandos do *backend* e enviar para os atuadores.
  - ***Redis - Celery Broker*:** É responsável pelo armazenamento em memória de dados que o *Celery* usa como uma fila de mensagens de alta *performance*. Ele armazena as tarefas que serão processadas no *backend*, garantindo que um *worker* as realize.
- ***Backend*:**
  - **API/Cliente MQTT:** Esta API expõe os *endpoints* que o sistema irá interagir (ex: buscar históricos, ligar atuador, agendar comandos).
  - ***Tasks Celery*:** São funções responsáveis por encapsular a lógica de processamento de dados e envio de comandos para posteriormente serem executadas em segundo plano.

  - **Banco de Dados:** Onde todos os dados são armazenados, incluindo dados dos sensores, atuadores e informações do usuário.

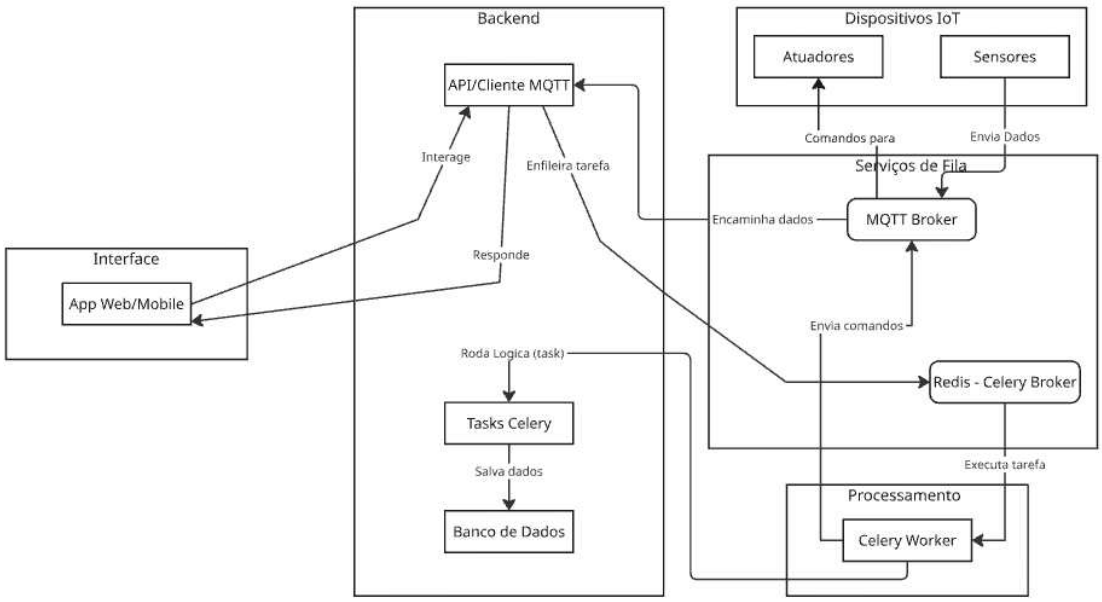
- **Processamento:**
  - ***Celery Worker*:** É o trabalhador que escuta o *Redis* (*Celery Broker*), recebe as tarefas que estão disponíveis na fila e as executa. Ele é responsável por executar as "*Tasks Celery*" em segundo plano.
- **Interface:**
  - ***APP Web/Mobile*:** Trata-se da interface onde os usuários interagem com o sistema. Através dela pode-se configurar os dispositivos, visualizar dados e realizar comandos.

Fluxos de dados e comandos:

1. **Dados dos Sensores:**
   - Os sensores coletam os dados e enviam para o MQTT *Broker*.
   - O MQTT *Broker* encaminha esses dados para a API/Cliente MQTT no *backend Django*.
   - O *backend* enfileira a tarefa desses dados no *Redis Celery Broker*.
   - O *Celery Worker* executa a tarefa do *Redis*, que roda a lógica (*Task*), e essa lógica salva os dados no Banco de Dados.
2. **Fluxo de Interação do Usuário:**
   - O *App Web/Mobile* interage com a API do *Backend* solicitando dados do histórico, ou aplicando comandos.
   - O *backend* responde o *App Web/Mobile* com os dados esperados.
3. **Comandos para Atuadores:**
   - Quando um comando desse é enviado (Por uma requisição do *App Web/Mobile* ou uma tarefa agendada), o *Celery Worker* executa uma *task* que decide o comando e envia para o MQTT *Broker*.
   - O MQTT *Broker* envia o comando para os atuadores IoT correspondentes, que executam a ação física.

Para realizar a comunicação *backend*, sensores e *broker* MQTT, foram utilizados os protocolos *Publish/Subscribe*, responsável por criar tópicos onde é possível se comunicar entre os sensores e a aplicação *backend*, e o protocolo *Producer/Consumer*, que utiliza o *broker Redis* responsável por executar operações em fila, o que torna a aplicação mais rápida e escalável.

Figura 6 – Diagrama de Fluxo geral do sistema AgroInfo V2



Fonte: Elaborado pelo autor.

4.4 Modelagem do Sistema

4.4.1 Diagrama de Casos de Uso

Com o objetivo de melhorar a visão das ações dos usuários no sistema, foi desenvolvido o diagrama de casos de uso na Figura **??**. Contamos com a representação de 2 atores: primeiramente, o usuário do sistema, que poderá acessar as funcionalidades do sistema responsáveis por monitorar as variáveis ambientais e os registros dos dados, como monitorar a temperatura, monitorar a umidade, acessar histórico de dados, ligar/desligar sistema de irrigação. O segundo trata-se do sistema em nuvem, o *broker* MQTT, que será responsável por intermediar a comunicação entre os sensores e o aplicativo, postando os dados captados pelos sensores, identificando falhas nos sensores caso ocorra, além de postar os dados que serão armazenados no banco de dados (*SQLite3*) posteriormente.

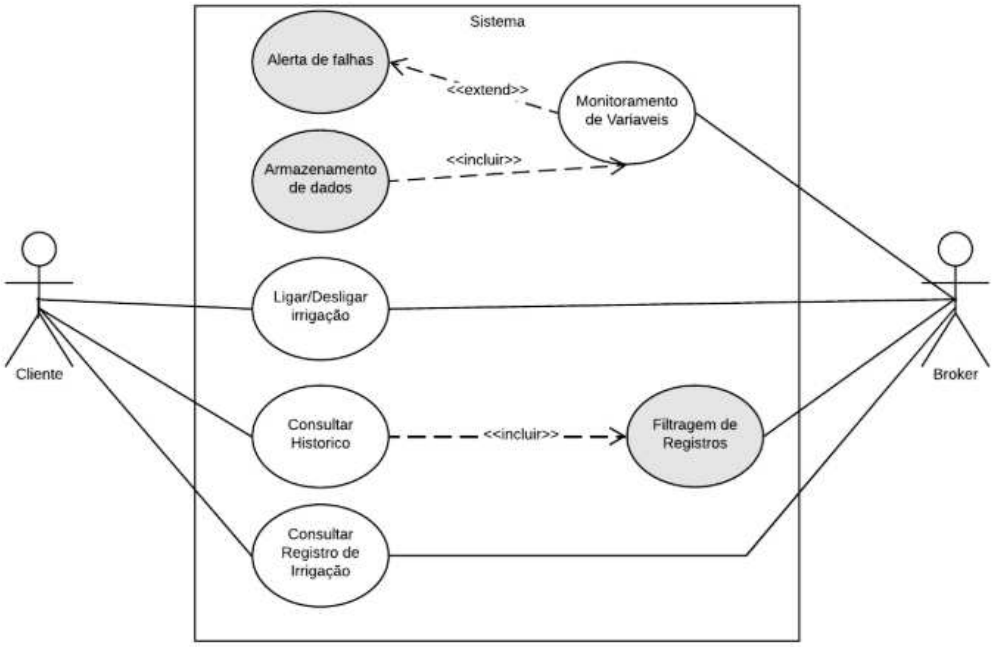
O diagrama de casos de uso demonstra as funcionalidades do sistema que promovem a interação dos usuários do sistema, podendo haver dependências entre eles para visualizar a lógica da aplicação.

- **Ligar/Desligar irrigação:** Este caso mostra a funcionalidade que possibilita o usuário realizar a ação de ligar ou desligar o sistema de irrigação.
- **Consultar Histórico:** Este caso demonstra o cenário em que os usuários podem acessar o

histórico de dados que foram detectados pelos sensores.

- **Consultar Registro de irrigação:** Este caso descreve o cenário onde o usuário tem acesso ao registro de todas as requisições de ligar e desligar o sistema de irrigação.
- **Filtragem de Registro:** Este caso mostra o cenário onde o usuário pode realizar a filtragem por data/hora dos dados armazenados no banco de dados do sistema.
- **Alerta de falhas:** Este cenário mostra o caso onde há uma falha nos sensores é identificada pelo *broker* MQTT, o sistema envia uma notificação de aviso.
- **Armazenamento de dados:** Este caso mostra o cenário que os dados postados pelo *broker* são armazenados no banco de dados do sistema.

Figura 7 - Diagrama de casos de uso



Fonte: (Elaborado pelo autor).

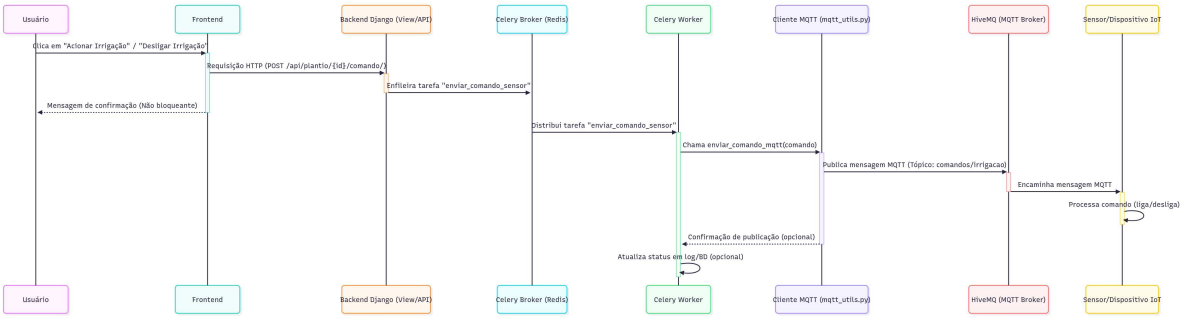
4.4.2 Diagrama de sequência comando para sensor (Acionar /desligar Irrigação)

Para auxiliar na visualização da interação do usuário ao utilizar o sistema, foram criados dois diagramas de sequência para representar o fluxo de dados entre os componentes. Neste diagrama, demonstramos o fluxo de dados para o envio de comandos para o sensor.

Na interface do sistema, o usuário seleciona “Acionar Irrigação” ou “Desligar Irrigação”. Seguidamente, o *frontend* envia uma requisição *HTTP* do tipo *POST* para a API do *backend*. Após receber essa requisição, em vez de processar diretamente o comando, o *backend*

aciona uma tarefa chamada ‘`enviar_comando_sensor`‘ para o *Celery Broker* (*Redis*). Um *Celery Worker* disponível executa essa tarefa que chama ‘`enviar_comando_mqtt()`‘ do cliente MQTT. Esta função irá publicar a mensagem MQTT com o comando escolhido (ligar ou desligar) no tópico de comando. O *HiveMQ* leva esta mensagem MQTT para o Sensor/Dispositivo IoT correspondente, que havia se inscrito no mesmo tópico. O Dispositivo recebe a mensagem e executa a ação que foi definida pelo usuário.

Figura 8 – Diagrama de sequência com comando para sensor



Fonte: Elaborado pelo autor.

4.4.3 Verificação Periódica de Status e Alertas do sensor

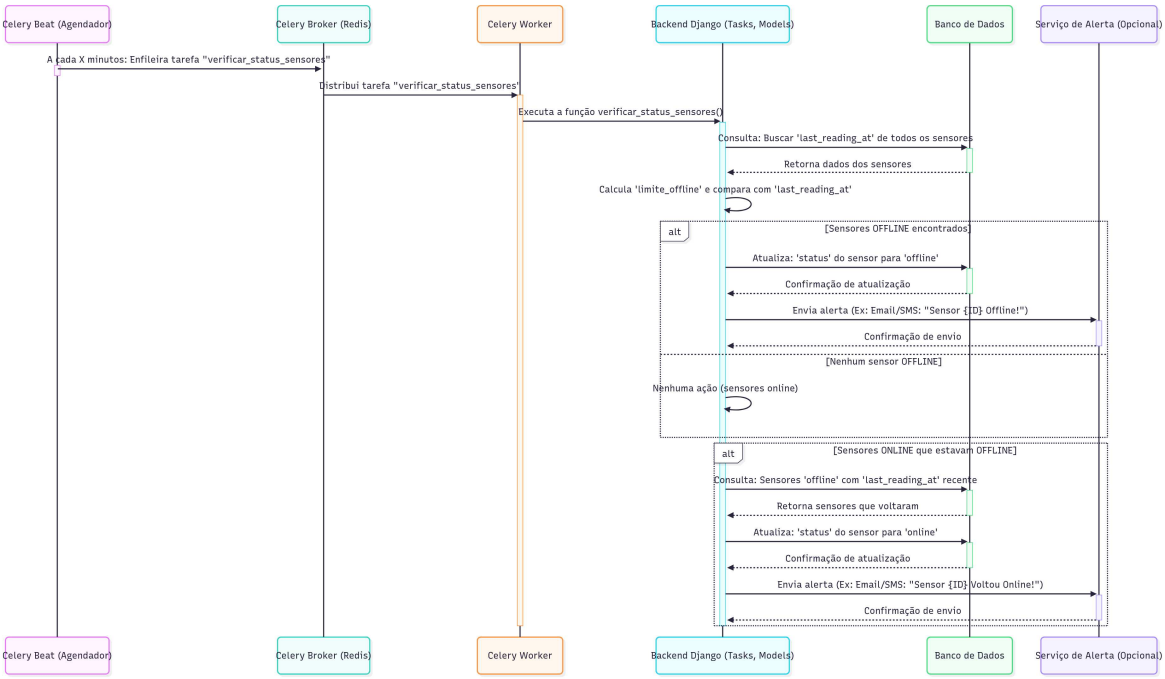
No processo da funcionalidade de Verificação periódica dos sensores e alertas, criamos um diagrama de sequência para sua visualização detalhada. Neste processo, o *Celery Beat* desempenha o papel de agendador de tarefas, enfileirando a tarefa ‘`verificar_status_sensores`‘ no *Celery Broker* (*Redis*) em intervalos regulares. Isso torna a verificação automática, sem necessidade de intervenção manual.

O *Celery Worker* está sempre aguardando tarefas na fila para processar. Ele pega a tarefa ‘`verificar_status_sensores`‘ do *Celery Broker*. Quando a executa, utiliza a lógica contida no *backend* e então realiza a consulta no banco de dados para obter a última leitura dos sensores cadastrados. Com essa informação, o *backend* calcula um limite *offline* (nesse sistema foi definido o tempo de 10 minutos) e assim determina se o sensor enviou dados recentemente.

Caso o sistema identifique que os sensores não reportam dados desde o limite *offline* e ainda estão identificados como *online*, o *backend* atualiza o *status* de *online* para *offline* no banco de dados. Em um fluxo adicional, o sistema também verifica se algum sensor que estava antes marcado como *offline* retornou a enviar dados; nesse caso, o sistema atualiza o banco de dados para o *status* de *online*. Após esse processo, o sistema finaliza as atividades relacionadas a

essas tarefas e aguarda a próxima atividade agendada pelo *Celery Beat*.

Figura 9 – Diagrama de sequência Verificação Periódica de Status e Alertas do sensor



Fonte: Elaborado pelo autor.

4.4.4 Diagrama de Classe

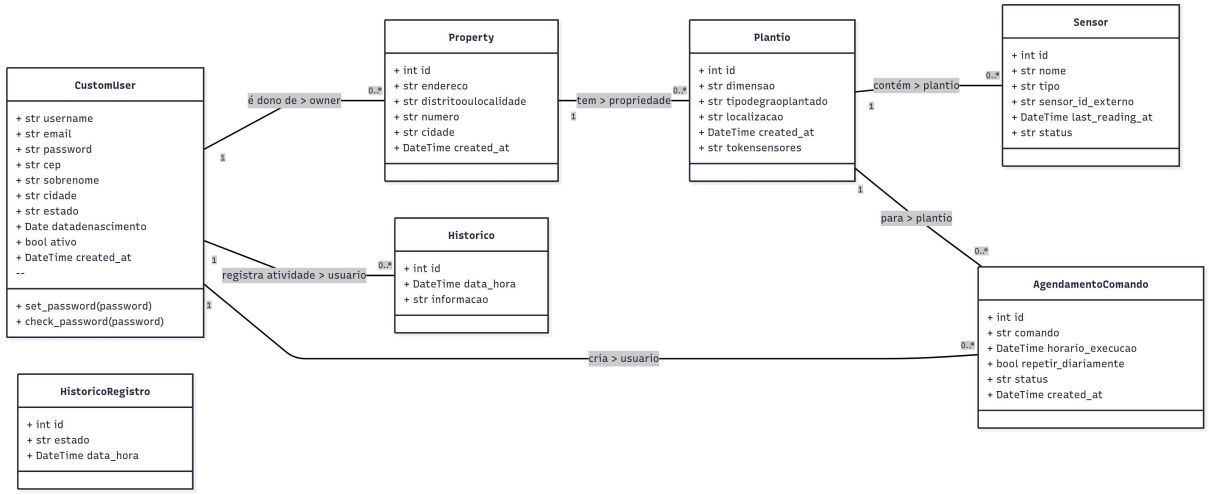
O diagrama da Figura 10 é uma representação da estrutura das funcionalidades a serem adicionadas ao sistema, mostrando visualmente as classes, relacionamentos e associações, assim fornecendo uma visão estática do sistema.

Definição das classes:

- **MVT:** A sigla MVT não é uma classe e sim uma arquitetura. A seção 4.5.2 aborda isso em detalhes.
- ***Plantio*:** É utilizada para criar objetos de plantio e possui uma chave estrangeira que referencia o objeto *Property*, que é ponto de acesso para seus métodos.
- ***Property*:** É utilizada para criação de objetos no banco de dados. Possui uma chave estrangeira para o objeto *User*, que torna possível acessar seus métodos.
- ***User*:** É utilizada para criação de objetos de usuário.
- ***Sensor*:** É utilizado para a criação de objetos de sensores no banco de dados. Possui uma chave estrangeira que referencia um objeto *Plantio* ao qual tem relação.

- ***Histórico*:** É utilizado para criação de objetos de histórico geral da aplicação no banco de dados, possui uma chave estrangeira para um (*CustomUser*), armazenando quem realizou a tarefa.
- ***HistóricoDeRegistro*:** É utilizada para criação de objetos de registro de comandos no banco de dados. Registra cada vez que o comando ligar/desligar a irrigação é enviada.
- ***AgendamentoDeComando*:** É utilizado para criação de objetos de agendamentos de comandos no banco de dados, tornando possível programar o agendamento para ligar/-desligar o sistema de irrigação automaticamente em horários específicos. Possui chaves estrangeiras para um objeto *CustomUser* e para um objeto *Plantio*, especificando o local da ação.

Figura 10 – Diagrama de Classe



Fonte: Elaborado pelo autor.

4.5 Tecnologias e Ferramentas Utilizadas

4.5.1 Ferramentas Utilizadas

Para o desenvolvimento do sistema, foram selecionadas as seguintes tecnologias e ferramentas:

- **Linguagem de Programação:** *Python*, devido à sua flexibilidade e vasta disponibilidade de bibliotecas para desenvolvimento *web* e IoT.
- ***Frameworks*:** *Django* e *Django REST Framework*, para implementação do *backend* e exposição de APIs *REST*.

- **Protocolo de Comunicação:** MQTT (*Message Queuing Telemetry Transport*), por ser leve e eficiente na transmissão de dados entre sensores e servidores.
- **Broker de Mensagens para o Celery:** Utilizado com *Broker* de mensagens para o *Celery*. O *Redis* atua como uma fila de alta *performance*, onde as tarefas são armazenadas temporariamente antes de serem executadas.
- **Sistema de Filas de Tarefas:** O *Celery* é essencial para execução em segundo plano das tarefas enfileiradas pelo *Redis*.
- **Agendador de Tarefas Periódicas:** Uma extensão do *Celery* que permite o agendamento de tarefas periódicas diretamente a partir do banco de dados do sistema.
- **Banco de Dados:** *SQLite3*, utilizado para armazenar as leituras ambientais e *logs* do sistema.
- **Plataforma de Hospedagem:** Inicialmente, o sistema será executado em ambiente local, com possibilidade de migração para servidores em nuvem.

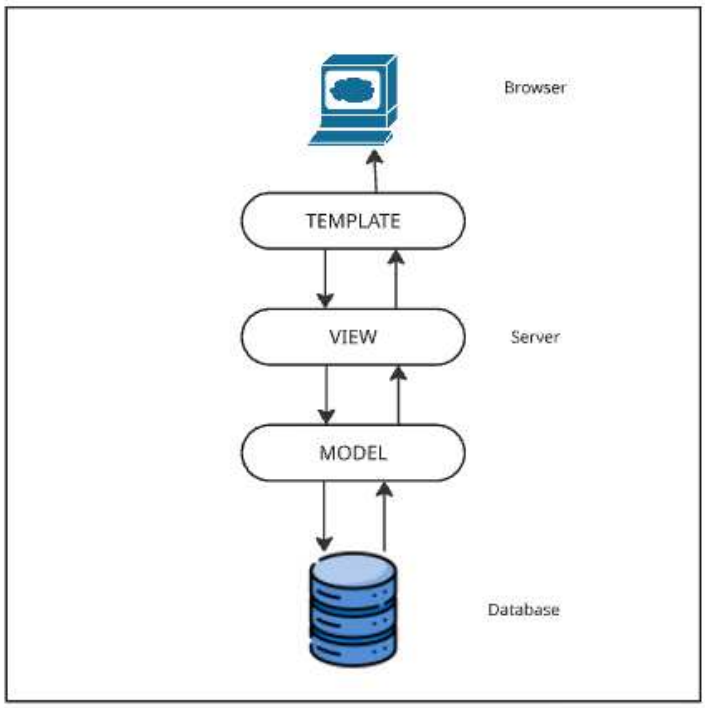
4.5.2 Arquitetura do Backend

Para o desenvolvimento desse projeto, foi utilizado o padrão MVT (*Model, Template, View*), que se trata de uma variação do padrão *MVC* (*Model, View, Controller*), mais adaptada para a utilização do *Django*, *framework* que foi utilizado neste trabalho.

É baseado em separações em três camadas que estão conectadas e desempenham papéis diferentes na aplicação.

- ***Template*:** Responsável por apresentar a parte visual e demonstrar dados, onde os arquivos `HTML` serão renderizados.
- ***View*:** É onde os dados serão formatados, exibindo a interface e as informações contidas no *Model*.
- ***Model*:** Onde é realizada a lógica para estruturar os dados, manipula os dados que serão coletados pelo *broker*, além de mapear o banco de dados.

Figura 11 - Representação esquematica do MVT



Fonte: Elaborado pelo autor.

4.5.3 Broker MQTT

O *broker* MQTT utilizado no desenvolvimento da aplicação trata-se do sistema disponibilizado pela *HiveMQ*. Neste projeto, foi utilizado o plano gratuito. Ao criar uma conta no *cluster*, é possível ter acesso às informações que são essenciais para conectar o *broker* para clientes, produtores e consumidores. A plataforma fornece as informações de conexão como a *URL* de acesso. Como a plataforma *HiveMQ* é disponibilizada via *AWS*, a *URL* é utilizada por clientes produtores e clientes consumidores para terem acesso ao *broker*.

A segurança da comunicação é garantida pela utilização de *URLs* de *TLS* (*Transport Layer Security*) para MQTT e *WebSocket*, o que garante a segurança por meio de criptografia dos dados de comunicação, protegendo as informações enquanto viajam entre os dispositivos e o *broker*.

Para iniciar o recebimento de dados e ativar o *broker*, é necessário criar credenciais. A própria plataforma *HiveMQ* pode criar de forma automática as credenciais necessárias para a aplicação. Uma vez que o *broker* está ativo com as credenciais, o próximo passo é definir os tópicos. Os tópicos servem para separar os dados dos produtores, garantindo que cada um receba as suas informações pertinentes. O nome dos tópicos é definido por um *serial token* obtido pelo usuário, de forma que cada usuário terá acesso ao seu próprio tópico para recebimento dos dados coletados e enviados pelos sensores. O *backend* da aplicação realiza a tarefa de coletar e salvar os dados no banco de dados, considerando os dados e o tópico referente ao usuário que está conectado à aplicação.

Um aspecto importante do *HiveMQ* é o *Quality of Service* (*QoS*) das mensagens, que se trata de um sistema de níveis que garantem a entrega das mensagens. Nos níveis 1 e 2, o *broker* rastreia as mensagens não confirmadas e as armazena em uma fila, garantindo a entrega mesmo que haja interrupções. No nível 0, não há rastreamento ou armazenamento de dados após uma perda de conexão, e neste nível as mensagens podem ser descartadas.

Quando o *broker* entra em execução e os tópicos estão definidos, o *HiveMQ* recebe os dados enviados pelos sensores e armazena as principais informações como o conteúdo da mensagem, os níveis de garantia (*QoS*) e o momento em que a mensagem foi recebida (*Timestamp*). Essas informações ficam disponíveis aos consumidores.

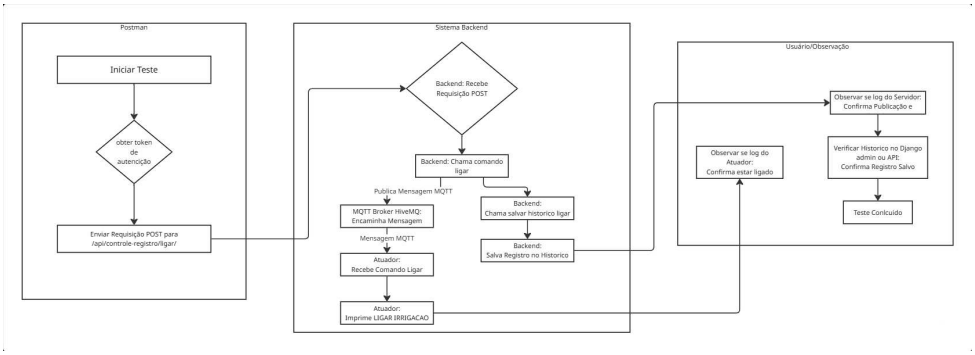
4.5.4 Softwares utilizados

Nesta seção, detalharemos os *softwares* utilizados na aplicação, a configuração utilizada e como foram realizados os testes.

4.5.4.1 Postman

Os testes das funcionalidades do sistema foram realizados no *Postman* em suas configurações padrões. A plataforma funciona como uma API que permite publicar, consumir e gerenciar APIs, podendo simular as requisições que um *frontend* geraria. Nesta aplicação, o *Postman* foi utilizado para enviar requisições e receber as respostas das APIs e verificar se eram as esperadas. Na ferramenta, foi definido o método utilizado (`POST` ou `GET`), depois informamos a *URL* da API a ser testada e o formato da entrada de dados *JSON*, quando necessário haver uma entrada. O sistema é desenvolvido em *Django REST*. Na Figura 13, visualizamos a execução. Ao clicar em "*Send*", a API retorna os dados que são exibidos no *Body*. Na Figura 12, podemos ver o diagrama que mostra a relação entre o programa de teste e o sistema *backend*.

Figura 12 – Diagrama de Testes



Fonte: Elaborado pelo autor.

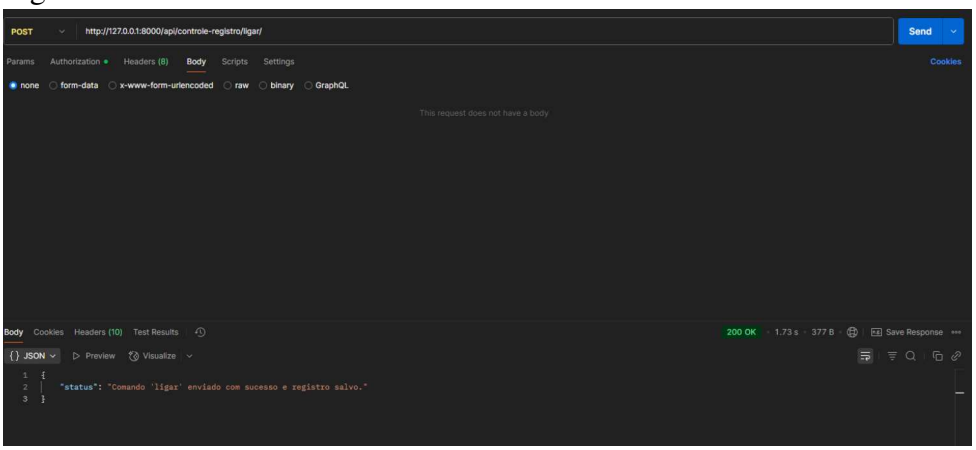
4.5.5 Hardware Utilizado

Nesta seção, detalharemos o *hardware* utilizado para o desenvolvimento da aplicação e para a realização dos testes.

4.5.5.1 Computador

Para o desenvolvimento da aplicação do *backend* e para a realização dos testes, foi utilizado um computador com as seguintes especificações:

Figura 13 – Interface do software Postman



Fonte: Elaborado pelo autor.

- **Sistema Operacional:** *Windows 11 Home Single Language 64 bits* Versão 22H2.
- **Processador:** *AMD Ryzen 5 3600 3.6GHz.*
- **Memória RAM:** 16,0*GB*.

4.6 Testes e Validação

Os resultados dos testes que foram realizados para avaliar o grau de desenvolvimento serão classificados desta maneira:

- **Sucesso:** A API executou e respondeu de forma esperada.
- **Falha Média:** A API executou, mas respondeu de forma incorreta.
- **Falha Alta:** A API executou, mas não houve resposta.
- **Falha Crítica:** A API não executou.

4.6.1 Cronograma dos Testes

Este cronograma mostra o período aproximado em que as atividades de planejamento e aplicação deverão ser realizadas.

Tabela 4 – Cronograma 2025

Atividades	**Março**	**Abril**	**Maio**	**Junho**	**Julho**
Realização do teste de Unidade	X	X	X		
Planejamento do documento de testes		X	X		
Especificação do caso de testes			X	X	
Realização do teste funcional			X	X	
Realização do teste de Integração				X	
Realização do teste de Desempenho					X

Fonte: Elaborada pelo autor.

4.6.2 Funcionalidades a serem testadas

Nesta seção, detalharemos as funcionalidades a serem testadas no sistema. Cada funcionalidade terá uma breve descrição, sua identificação, nome do caso de teste e a qual requisito a funcionalidade está relacionada.

Tabela 5 – Tabela de Casos de Teste

Identificação	**Nome da Funcionalidade**	**Requisito Funcional**	**Descrição**
CT01	Alerta de Falhas	RF001	Verificar se o sensor está enviando dados.
CT02	Ligar registro	RF002	Enviar o comando ligar para o Atuador.
CT03	Desligar registro	RF002	Enviar o comando desligar para o Atuador.
CT04	Agendar atuador	RF002	Verificar o agendamento para ativação dos atuadores.
CT05	Filtragem de Histórico por comando desligar	RF003	Realizar filtragem nos dados do histórico por comando “desligar”.
CT06	Filtragem de Histórico por data de início	RF003	Realizar filtragem nos dados do histórico por data de início.
CT07	Filtragem de Histórico por data final	RF003	Realizar filtragem nos dados do histórico por última data registrada.
CT08	Filtragem de Histórico por combinação de filtros	RF003	Realizar filtragem nos dados do histórico por comando e data combinados.
CT09	Filtragem de Histórico por comando ligar	RF003	Realizar filtragem nos dados do histórico por comando “ligar”.
CT10	Histórico de Registro	RF004	Salvar dados de comando ligar/-desligar no banco de dados.
CT11	Cadastrar Sensor	RF005	Criar um objeto do tipo sensor.
CT12	Cadastrar Atuador	RF006	Criar um objeto do tipo Atuador.

Fonte: Elaborada pelo autor.

4.7 Recursos necessários

Para a realização dos testes, é necessária a utilização de *hardware*, *software* e sistema de testes. Para uma melhor avaliação dos testes, especificamos o ambiente de execução e os recursos utilizados.

Tabela 6 – Tabela de Ferramentas e Recursos

Nome	**Justificativa**
Postman	A ferramenta é utilizada para simular um *frontend* e enviar entrada de dados.
VS Code	A ferramenta é necessária para a execução da aplicação.
Computador	O dispositivo é necessário para a execução de todas as ferramentas utilizadas nos testes.
Simulador de Sensor em *Python*	Atuará simulando um sensor enviando dados para o *broker*.

Fonte: Elaborada pelo autor.

5 RESULTADOS

5.1 Casos de Teste

Foram elaborados casos de teste onde foram enviadas entradas de dados que simulam um ambiente real onde se esperam o retorno de determinadas saídas correspondentes. Seguindo os passos determinados, os testes buscam imitar um cenário de produção onde o usuário do sistema realizará suas tarefas. Com base no sistema desenvolvido, foram realizados testes para avaliar as saídas obtidas pelas APIs da aplicação.

Tabela 7 – Caso de teste 01 - Ligar registro

Caso de Teste 01	Ligar registro
Descrição	Verificar se o endpoint api/controle-registro/ligar/ está enviando o comando ligar/
Funcionalidade	RF1
Pré condição	• É necessário realizar o login no admin. • O servidor está iniciado. • É necessário ter um cadastro de uma propriedade e de um plantio. • A task verificar_status_sensores deve estar implementada

Passos	1. Abrir o Django admin:
	  • Fazer login como superusuário
	  • Definir a las_reading 30s segundo no passado
	  • Definir o status como online
	2. Salvar

Resultados Esperados	1 – Exibição da mensagem no terminal “ALERTA: Sensor [ID DO SENSOR] - [NOME DO SENSOR] está OFFLINE!”.

Fonte: Elaborada pelo autor.

Tabela 8 – Caso de teste 02 - Ligar registro

Caso de Teste 02	Ligar registro
Descrição	Verificar se o endpoint api/controle-registro/ligar/ está enviando o comando ligar/
Funcionalidade	RF2
Pré condição	• É necessário realizar o login no sistema. • O servidor está iniciado. • É necessário ter um cadastro de uma propriedade e de um plantio. • O endpoint api/controle-registro/ligar/ deve estar implementado.
Passos	1. Abrir o Postman e criar uma requisição com algumas requisições: • Método: POST • URL: http://127.0.0.1:8000/api/controle-registro/ligar/ • Headers: Content-type: application/json • Authorization: Bearer “token serial” 2. Enviar Requisição
Resultados Esperados	1. Exibição da mensagem no terminal do servidor HTTP_200_OK. 2. Recebimento no Postman em JSON o Body informado na requisição.

Fonte: Elaborada pelo autor.

Tabela 9 – Caso de teste 03 - Desligar registro

Caso de Teste 03	Desligar registro
Descrição	Verificar se o endpoint api/controle-registro/desligar/ está enviando o comando desligar/
Funcionalidade	RF2
Pré condição	• É necessário realizar o login no sistema. • O servidor está iniciado. • É necessário ter um cadastro de uma propriedade e de um plantio. • O endpoint api/controle-registro/desligar/ deve estar implementado.
Passos	1. Abrir o Postman e criar uma requisição com algumas requisições: • Método: POST • URL: http://127.0.0.1:8000/api/controle-registro/desligar/ • Headers: Content-type: application/json • Authorization: Bearer “token serial” 2. Enviar Requisição
Resultados Esperados	1. Exibição da mensagem no terminal do servidor HTTP_200_OK. 2. Recebimento no Postman em JSON o Body informado na requisição.

Fonte: Elaborada pelo autor.

Tabela 10 – Caso de teste 04 - Agendar registro

Caso de Teste 04	Agendar registro
Descrição	Verificar se o endpoint /api/agendamentos/ está enviando o comando para agendamento./
Funcionalidade	RF2
Pré condição	• É necessário realizar o login no sistema. • O servidor está iniciado. • É necessário ter um cadastro de uma propriedade e de um plantio. • O endpoint /api/agendamentos/ deve estar implementado.
Passos	1. Abrir o Postman e criar uma requisição com algumas requisições: • Método: POST • URL: http://127.0.0.1:8000/api/agendamentos/ • Headers: Content-type: application/json • Authorization: Bearer “token serial” • Body: “plantio”: “2”, “comando”: “ligar”, “horario_execucao”: “2025-06-30T12:00:00Z”, “repetir_diariamente”: “false” 2. Enviar Requisição
Resultados Esperados	1. Exibição da mensagem no terminal do servidor HTTP_201_CREATED. 2. Recebimento no Postman em JSON o Body informado na requisição.

Fonte: Elaborada pelo autor.

Tabela 11 – Caso de teste 05 - Filtragem de Histórico por comando desligar

Caso de Teste 05	Filtragem de Histórico por comando desligar
Descrição	Verificar se o endpoint /api/historico_comandos/?estado=desligar está enviando o comando está respondendo os dados corretamente.
Funcionalidade	RF3
Pré condição	• É necessário realizar o login no sistema. • O servidor está iniciado. • É necessário ter um cadastro de uma propriedade e de um plantio. • O endpoint api/controle-registro/?estado=<valor>/ deve estar implementado.

Passos	1. Abrir o Postman e criar uma requisição com algumas requisições:
	  • Método: GET
	  • URL: http://127.0.0.1:8000/api/historico_ comandos/?estado=desligar
	  • Headers: Content-type: application/json
	  • Authorization: Bearer “token serial”
	2. Enviar Requisição

Resultados Esperados	1. Exibição da mensagem no terminal do servidor HTTP_200_OK. 2. Recebimento no Postman em JSON uma lista de mensagens.

Fonte: Elaborada pelo autor.

Tabela 12 – Caso de teste 06 - Filtragem de Histórico por data de início

Caso de Teste 06	Filtragem de Histórico por data de início
Descrição	Verificar se o endpoint /api/historico_comandos/?data_inicio=<data-início> está enviando o comando está respondendo os dados corretamente.
Funcionalidade	RF3
Pré condição	• É necessário realizar o login no sistema. • O servidor está iniciado. • É necessário ter um cadastro de uma propriedade e de um plantio. • O endpoint http://127.0.0.1:8000/api/historico_comandos/?data_inicio=<data-início> deve estar implementado.
Passos	1. Abrir o Postman e criar uma requisição com algumas requisições: • Método: GET • URL:http://127.0.0.1:8000/api/historico_comandos/?data_inicio=2025-06-20 • Headers: Content-type: application/json • Authorization: Bearer “token serial” 2. Enviar Requisição
Resultados Esperados	1. Exibição da mensagem no terminal do servidor HTTP_200_OK. 2. Recebimento no Postman em JSON uma lista de mensagens.

Fonte: Elaborada pelo autor.

Tabela 13 – Caso de teste 07 - Filtragem de Histórico por data final

Caso de Teste 07	Filtragem de Histórico por data final
Descrição	Verificar se o endpoint /api/historico_comandos/?data_final=<data-final> está enviando o comando está respondendo os dados corretamente.
Funcionalidade	RF3
Pré condição	• É necessário realizar o login no sistema. • O servidor está iniciado. • É necessário ter um cadastro de uma propriedade e de um plantio. • O endpoint /api/historico_comandos/?data_final=<data-final> deve estar implementado.
Passos	1. Abrir o Postman e criar uma requisição com algumas requisições: • Método: GET • URL: http://127.0.0.1:8000/api/historico_comandos/?data_fim=2025-07-09 • Headers: Content-type: application/json • Authorization: Bearer “token serial” 2. Enviar Requisição
Resultados Esperados	1. Exibição da mensagem no terminal do servidor HTTP_200_OK. 2. Recebimento no Postman em JSON uma lista de mensagens.

Fonte: Elaborada pelo autor.

Tabela 14 – Caso de teste 08 - Filtragem de Histórico por combinação de filtros

Caso de Teste 08	Filtragem de Histórico por combinação de filtros
Descrição	Verificar se o endpoint /api/historico_comandos/?estado=<valor>&data_inicio=<data-inicio>&data_fim=<data-fim> está enviando o comando está respondendo os dados corretamente.
Funcionalidade	RF3
Pré condição	• É necessário realizar o login no sistema. • O servidor está iniciado. • É necessário ter um cadastro de uma propriedade e de um plantio. • O endpoint /api/historico_comandos/?estado=<valor> &data_inicio=<data-inicio>&data_fim=<data-fim> deve estar implementado.
Passos	1. Abrir o Postman e criar uma requisição com algumas requisições: • Método: GET • URL: http://127.0.0.1:8000/api/historico_ comandos/?estado=ligar& data_inicio=2025-06-25& data_fim=2025-07-09 • Headers: Content-type: application/json • Authorization: Bearer “token serial” 2. Enviar Requisição
Resultados Esperados	1. Exibição da mensagem no terminal do servidor HTTP_200_OK. 2. Recebimento no Postman em JSON uma lista de mensagens.

Fonte: Elaborada pelo autor.

Tabela 15 – Caso de teste 09 - Filtragem de Histórico por comando ligar

Caso de Teste 09	Filtragem de Histórico por comando ligar
Descrição	Verificar se o endpoint /api/historico_comandos/?estado=ligar está enviando o comando está respondendo os dados corretamente.
Funcionalidade	RF3
Pré condição	• É necessário realizar o login no sistema. • O servidor está iniciado. • É necessário ter um cadastro de uma propriedade e de um plantio. • O endpoint api/controle-registro/?estado=<valor>/ deve estar implementado.

Passos	1. Abrir o Postman e criar uma requisição com algumas requisições:
	  • Método: GET
	  • URL: http://127.0.0.1:8000/api/historico_ comandos/?estado=ligar
	  • Headers: Content-type: application/json
	  • Authorization: Bearer “token serial”
	2. Enviar Requisição

Resultados Esperados	1. Exibição da mensagem no terminal do servidor HTTP_200_OK. 2. Recebimento no Postman em JSON uma lista de mensagens.

Fonte: Elaborada pelo autor.

Tabela 16 – Caso de teste 10 - Histórico Registro

Caso de Teste 10	Histórico Registro
Descrição	Verifique se o endpoint /api/historico/ está cadastrando o sensor corretamente no banco de dados./
Funcionalidade	RF5
Pré condição	• É necessário realizar o login no sistema. • O servidor está iniciado. • É necessário ter um cadastro de uma propriedade e de um plantio. • O endpoint /api/historico/ deve estar implementado.
Passos	1. Abrir o Postman e criar uma requisição com algumas requisições: • Método: POST • URL: /api/sensores/cadastrar/ • Headers: Content-type: application/json • Authorization: Bearer “token serial” • Body: “comando”: “desligar”, “timestamp”: “2025-06-25T10:05:00Z”, “comando”: “ligar”, “timestamp”: “2025-06-25T10:00:00Z” 2. Enviar Requisição
Resultados Esperados	1. Exibição da mensagem no terminal do servidor HTTP_200_OK. 2. Recebimento no Postman em JSON o Body informado na requisição.

Fonte: Elaborada pelo autor.

Tabela 17 – Caso de teste 11 - Cadastrar Sensor

Caso de Teste 11	Cadastrar Sensor
Descrição	Verifique se o endpoint /api/sensores/cadastrar/ está cadastrando o sensor corretamente no banco de dados./
Funcionalidade	RF5
Pré condição	• É necessário realizar o login no sistema. • O servidor está iniciado. • É necessário ter um cadastro de uma propriedade e de um plantio. • O endpoint /api/sensores/cadastrar/ deve estar implementado.
Passos	1. Abrir o Postman e criar uma requisição com algumas requisições: • Método: POST • URL: /api/sensores/cadastrar/ • Headers: Content-type: application/json • Authorization: Bearer “token serial” • Body: “plantio”: “2”, “nome”: “Sensor de Umidade do Solo Sul”, “tipo”: “umidade”, “sensor_id_externo”: “UMIDADE-SUL-001” 2. Enviar Requisição
Resultados Esperados	1. Exibição da mensagem no terminal do servidor HTTP_201_CREATED. 2. Recebimento no Postman em JSON o Body informado na requisição.

Fonte: Elaborada pelo autor.

Tabela 18 – Caso de teste 12 - Cadastrar Atuador

Caso de Teste 12	Cadastrar Atuador
Descrição	Verifique se o endpoint /api/atuadores/cadastrar/ está cadastrando o atuador corretamente no banco de dados./
Funcionalidade	RF6
Pré condição	• É necessário realizar o login no sistema. • O servidor está iniciado. • É necessário ter um cadastro de uma propriedade e de um plantio. • O endpoint /api/atuadores/cadastrar/ deve estar implementado.
Passos	1. Abrir o Postman e criar uma requisição com algumas requisições: • Método: POST • URL: /api/atuadores/cadastrar/ • Headers: Content-type: application/json • Authorization: Bearer “token serial” • Body: “plantio”: “2”, “nome”: “Válvula Principal Setor Norte”, “tipo”: “irrigacao”, “atuador_id_externo”: “VALVULA-NORT-001” 2. Enviar Requisição
Resultados Esperados	1. Exibição da mensagem no terminal do servidor HTTP_201_CREATED. 2. Recebimento no Postman em JSON o Body informado na requisição.

Fonte: Elaborada pelo autor.

5.2 Relatório de teste

5.2.1 Introdução

Esta seção detalha e apresenta de forma relativamente próxima às falhas encontradas durante todo o período de desenvolvimento e testes na aplicação deste trabalho.

5.2.2 Funcionalidades testadas

Tabela 19 – Tabela de Resultados dos Casos de Teste

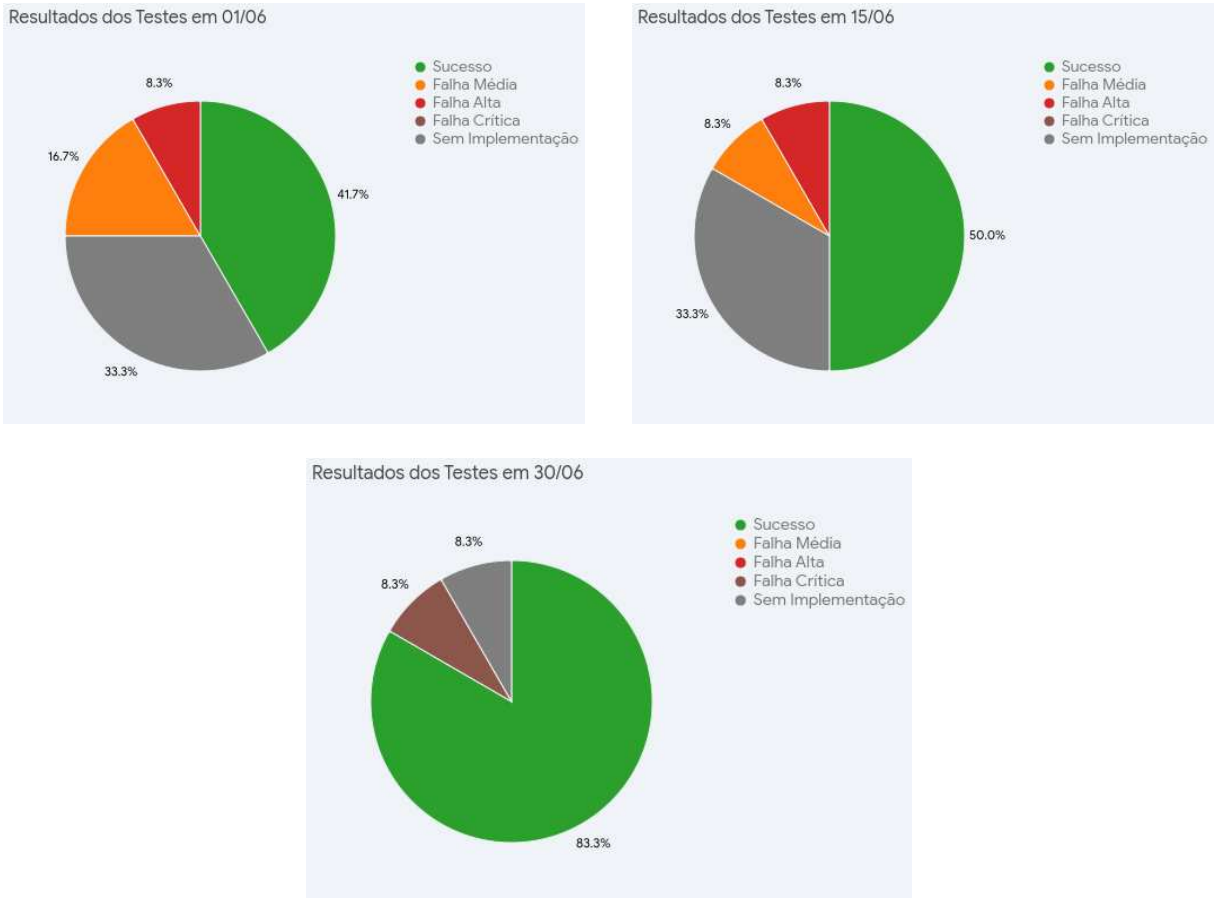
ID	**Caso de Teste**	**30/04**	**01/06**	**15/06**	**30/06**	**28/07**
CT01	Alerta de Falhas	Sem Impl.	Sem Impl.	Sem Impl.	Falha Crítica	Sucesso
CT02	Ligar registro	Sem Impl.	Sucesso	Sucesso	Sucesso	Sucesso
CT03	Desligar registro	Sem Impl.	Falha Média	Sucesso	Sucesso	Sucesso
CT04	Agendar registro	Sem Impl.	Sucesso	Sucesso	Sucesso	Sucesso
CT05	Filtragem Histórico por comando desligar	Sem Impl.	Sucesso	Sucesso	Sucesso	Sucesso
CT06	Filtragem Histórico por data início	Sem Impl.	Falha Média	Falha Média	Sucesso	Sucesso
CT07	Filtragem Histórico por data final	Sem Impl.	Sucesso	Sucesso	Sucesso	Sucesso
CT08	Filtragem Histórico por combinação filtros	Sem Impl.	Falha Alta	Falha Alta	Sucesso	Sucesso
CT09	Filtragem Histórico por comando ligar	Sem Impl.	Sucesso	Sucesso	Sucesso	Sucesso
CT10	Histórico de Registro	Sem Impl.	Sem Impl.	Sem Impl.	Sucesso	Sucesso
CT11	Cadastrar Sensor	Sem Impl.	Sem Impl.	Sem Impl.	Sucesso	Sucesso
CT12	Cadastrar Atuador	Sem Impl.	Sem Impl.	Sem Impl.	Sem Impl	Sucesso

Fonte: Elaborada pelo autor.

5.2.2.1 Visão geral dos resultados

Os gráficos a seguir mostram a visualização dos resultados aproximados dos testes, destacando os períodos em que houverem maiores taxas de sucesso. No período de 30/04/2025 e 28/07/2025 não foram incluídas pois constavam cem por cento.

Figura 14 – Gráficos dos resultados de testes

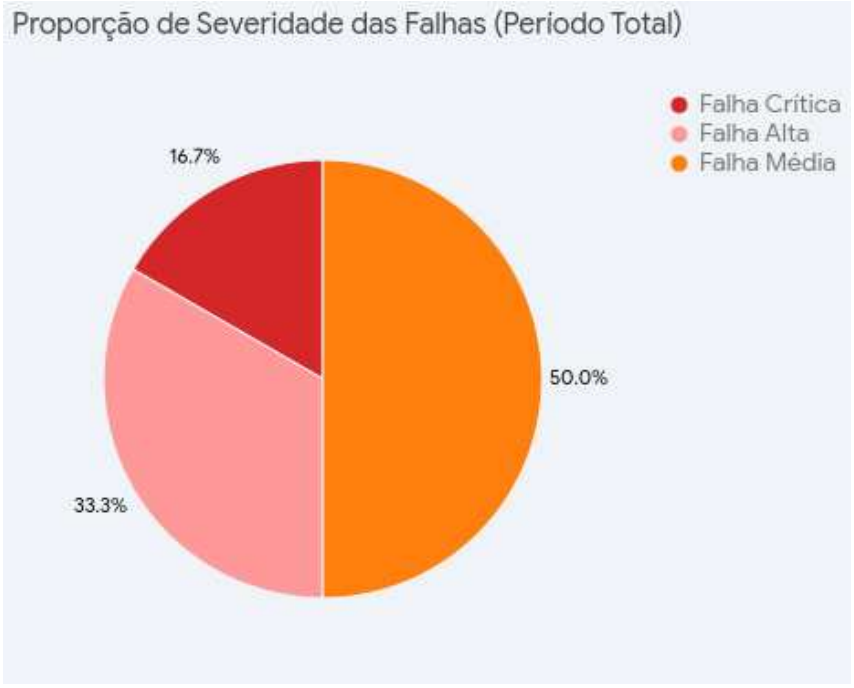


Fonte: Elaborada pelo autor.

5.2.2.2 Visão Geral da severidade das falhas

O gráfico a seguir mostra o grau de severidade das falhas ao decorrer de todo o período de testes.

Figura 15 – Gráfico da severidade das falhas



Fonte: Elaborada pelo autor.

6 CONCLUSÃO

Este trabalho busca aprimorar um sistema IoT existente, expandindo seu escopo para permitir, além do recebimento de dados de sensores, o envio de comandos para atuadores. Essa funcionalidade, essencial para uma maior automação e controle de plantios, é viabilizada por uma nova aplicação. Ela se conecta a um novo *broker*, responsável por fazer o processamento de dados em segundo plano e possibilitar a execução automática de ações em horários pré-definidos, conferindo assim uma capacidade autônoma ao sistema de irrigação.

Neste trabalho, observou-se que uma solução capaz de otimizar e automatizar propriedades agrícolas pode ser de baixo custo. Mesmo com uma aplicação simples, foi possível alcançar os resultados esperados, demonstrando a eficácia da abordagem. Diferente dos trabalhos relacionados, que por vezes se aplicam a resolver problemas específicos, este projeto mantém um escopo abrangente, capaz de solucionar desafios em diversas situações. A abrangência desta aplicação se dá por permitir, além da visualização dos dados transmitidos pelos sensores, a atuação remota do usuário, que pode enviar comandos aos atuadores, receber alertas de falhas (em caso de problemas de comunicação entre o sensor e o *broker*) e ter um maior gerenciamento dos dados armazenados no banco de dados.

Com o desenvolvimento de novas funcionalidades para o controle da automação foi significantemente aprimorado, aumentando assim a interação do usuário. O sistema agora possibilita ao usuário não somente obter as variáveis dos plantios, mas também realizar ações remotamente. Apesar dos avanços, podemos apontar algumas limitações que persistem no sistema atual. A principal delas é que, embora o sistema armazene os registros para todos os usuários, a dependência de banco de dados local pode ocasionalmente provocar a sobrecarga de dados ao longo do tempo. Para aumentar a escalabilidade, poderia haver o armazenamento desses dados na nuvem diretamente no *broker*; no entanto, essa solução ainda não foi implementada.

Outra limitação que permanece é a restrição do *broker* gratuito (*HiveMQ*), que somente permite a retenção de dados durante três dias e impede a configuração manual de tópicos, que são criados dinamicamente. Mesmo com o desenvolvimento do banco de dados para o armazenamento do histórico e do gerenciamento pelas aplicações, foi somente uma solução paliativa, que não eliminou a necessidade de um *broker* com mais flexibilidade, o que se torna um desafio técnico de infraestrutura do sistema.

Esta aplicação possibilita não somente analisar, mas também automatizar áreas agrícolas não somente no Vale do Jaguaribe, mas em diferentes regiões onde podem ser obtidas

variáveis de diferentes práticas agrícolas. O sistema atual deixou uma sólida base que possibilita a automação e controle de plantios, tendo em mente que outras regiões podem ter seus próprios sistemas de avaliação e gerenciamento de plantios, pode-se haver um aprimoramento da aplicação com base nas descobertas alcançadas neste trabalho.

Este trabalho abre espaço para futuras investigações e implementação de novas soluções que aumentem a eficiência e a escalabilidade. Pode ser realizada a migração de dados para a nuvem através do *broker* ou a implementação de banco de dados *NoSQL* (com *MongoDB*); que permitiria gerenciar de maneira escalavel o volume crescente de dados obtidos pelos sensores, o que eliminaria a preocupação com sobrecarga de dados, essa possibilidade poderiam ser explorada.

Outro ponto a ser melhorado seria a substituição do *broker* atualmente utilizado por um *broker* de código aberto com maior flexibilidade e escalabilidade (como o *Mosquitto*) que permite a configuração de dados ou soluções IoT na nuvem (*AWS IoT Core*, *Azure IoT*). Isso permitiria a retenção de dados por períodos mais longos.

O desenvolvimento de um módulo de *dashboard* personalizável que permitiria ao usuário criar seus próprios painéis para visualização de dados, com gráficos gerados conforme sua necessidade. Essas e outras possibilidades podem ser implementadas para aprimorar o sistema.

Em suma, este trabalho possibilita grandes benefícios para as propriedades agrícolas do Vale do Jaguaribe. Ele entrega aos usuários um sistema de baixo custo e fácil manutenção capaz de modernizar plantios, tornando-os mais eficientes. A aplicação se mostra promissora, mesmo com limitações que podem ser superadas futuramente.

REFERÊNCIAS

Berrios, C. e. a. H. Sistema iot baseado em esp32 para monitoramento em estufas agrícolas. **Revista de Tecnologias Sustentáveis**, v. 11, n. 3, p. 95–104, 2022.

Caligaris, M. A. e. a. **Desafios e oportunidades da agricultura brasileira frente às mudanças climáticas**. 2022. Disponível em: <https://www.embrapa.br/busca-de-publicacoes/-/publicacao/1144211>. Acesso em: 10 jun. 2025.

Carissimi, A.; Barbosa, D. **Computação em Nuvem**. [S.l.]: SENAI, 2016.

Carmo, J. H. **Frameworks para aplicações web em Python**. 2023. Trabalho de Conclusão de Curso – Universidade Federal do Ceará.

CEPEA. **PIB do agronegócio brasileiro cresceu 2,5% em 2023**. 2023. Disponível em: <https://www.cepea.esalq.usp.br/>. Acesso em: 10 jun. 2025.

CODEMEC. **IoT. Ou a Internet das Coisas**. 2016. <https://codemec.org.br/iot-ou-a-internet-das-coisas/>. [Online]. Acesso em: 17 de jul. de 2025.

Dantas, M. V. Desenvolvimento de apis rest com django. In: **Anais do Simpósio Brasileiro de Engenharia de Software (SBES)**. [S.l.: s.n.], 2023.

FAO. **O futuro da alimentação e da agricultura – Caminhos alternativos para 2050**. 2018. Disponível em: <https://www.fao.org/3/I8429PT/i8429pt.pdf>. Acesso em: 10 jun. 2025.

Godfray, H. C. J. e. a. Food security: The challenge of feeding 9 billion people. **Science**, v. 327, n. 5967, p. 812–818, 2010.

Golondrino, G. A. e. a. Sistema iot para monitoramento de variáveis climatológicas em agricultura urbana. **Revista Brasileira de Agricultura Urbana**, v. 2, n. 1, p. 40–50, 2022.

Gomes, J. S. e. a. Utilização da iot na agricultura sustentável. In: **Anais do Congresso Brasileiro de Agroinformática (SBIAgro)**. [S.l.: s.n.], 2023.

IBGE. **Censo Agropecuário 2017 – Resultados definitivos**. 2017. Disponível em: <https://censoagro2017.ibge.gov.br>. Acesso em: 10 jun. 2025.

Jesus, L. F. d. **Impactos da tecnologia da informação na transformação digital**. 2021. Disponível em: <https://repositorio.ifba.edu.br/jspui/handle/123456789/901>. Acesso em: 10 jun. 2025.

LENA, F. Q.; OLIVEIRA, A. M. d. **Utilização do Protocolo MQTT para Sistemas de IoT Voltado para Automação Residencial**. Santa Maria, RS, Brasil: [s.n.], 2018. 4 p. Disponível em: <https://www.tfgonline.lapinf.ufn.edu.br/media/midias/FabioLena.pdf>. Acesso em: 17 jul. 2025.

Muxito, E. e. a. Iot na agricultura – automação de pivôs e canais de irrigação com arduino e webservice. In: **Anais do Congresso de Engenharia Agrícola**. [S.l.: s.n.], 2018.

Planton. **Cloud Computing. Guia Completo para Iniciantes**. 2024. <https://blog.platon.com.br/cloud-computing-guia-completo-para-iniciantes/>. [Online]. Acesso em: 25 de jul. de 2025.

Santos, J. **AgroInfo: Sistema de monitoramento agrícola com Django e IoT**. 2024. Trabalho de Conclusão de Curso – Universidade Federal do Ceará.

Silva, J. P. d. e. a. Conceitos e aplicações da internet das coisas: uma revisão bibliográfica. **Revista de Informática Aplicada**, v. 6, n. 1, p. 24–35, 2020.