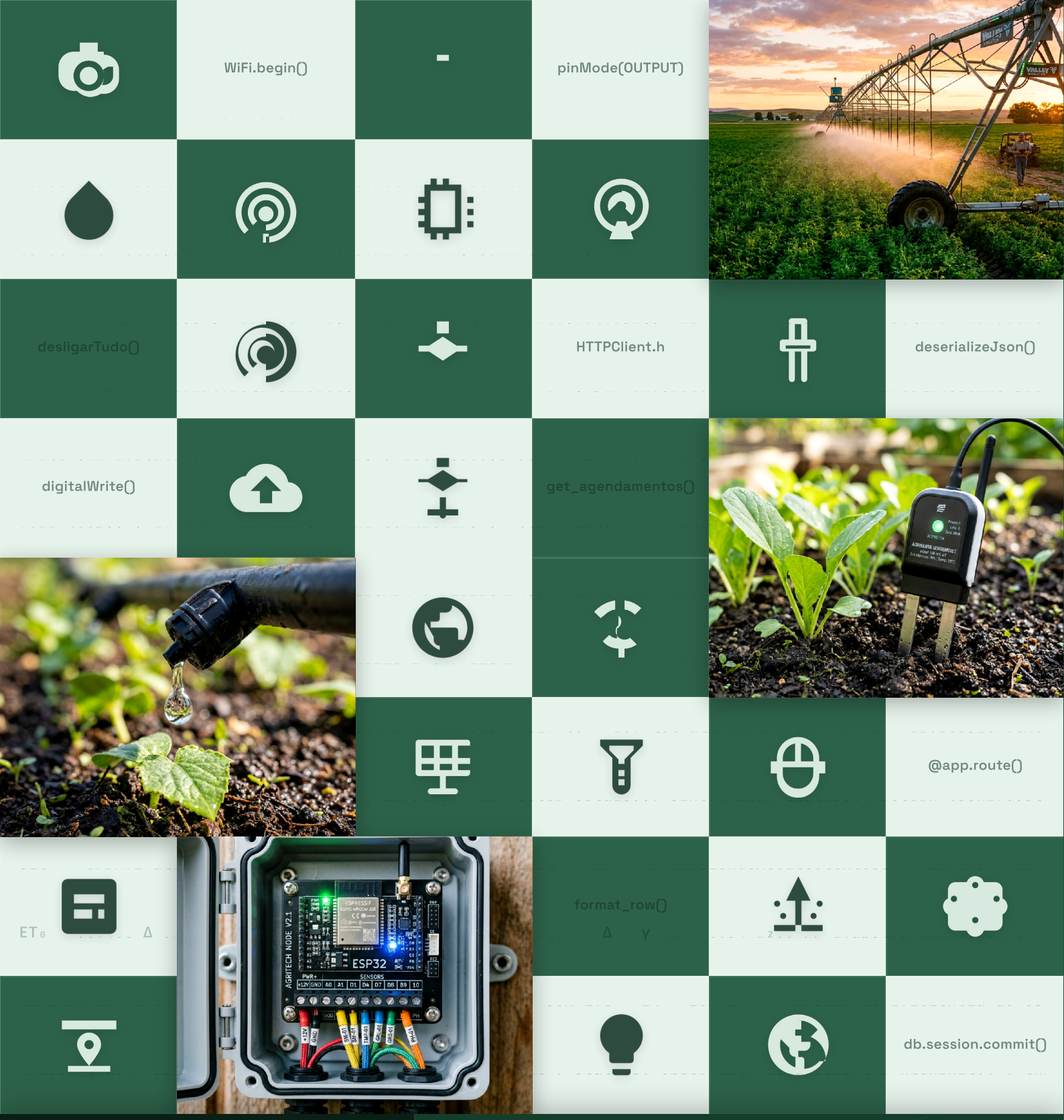


AUTORES

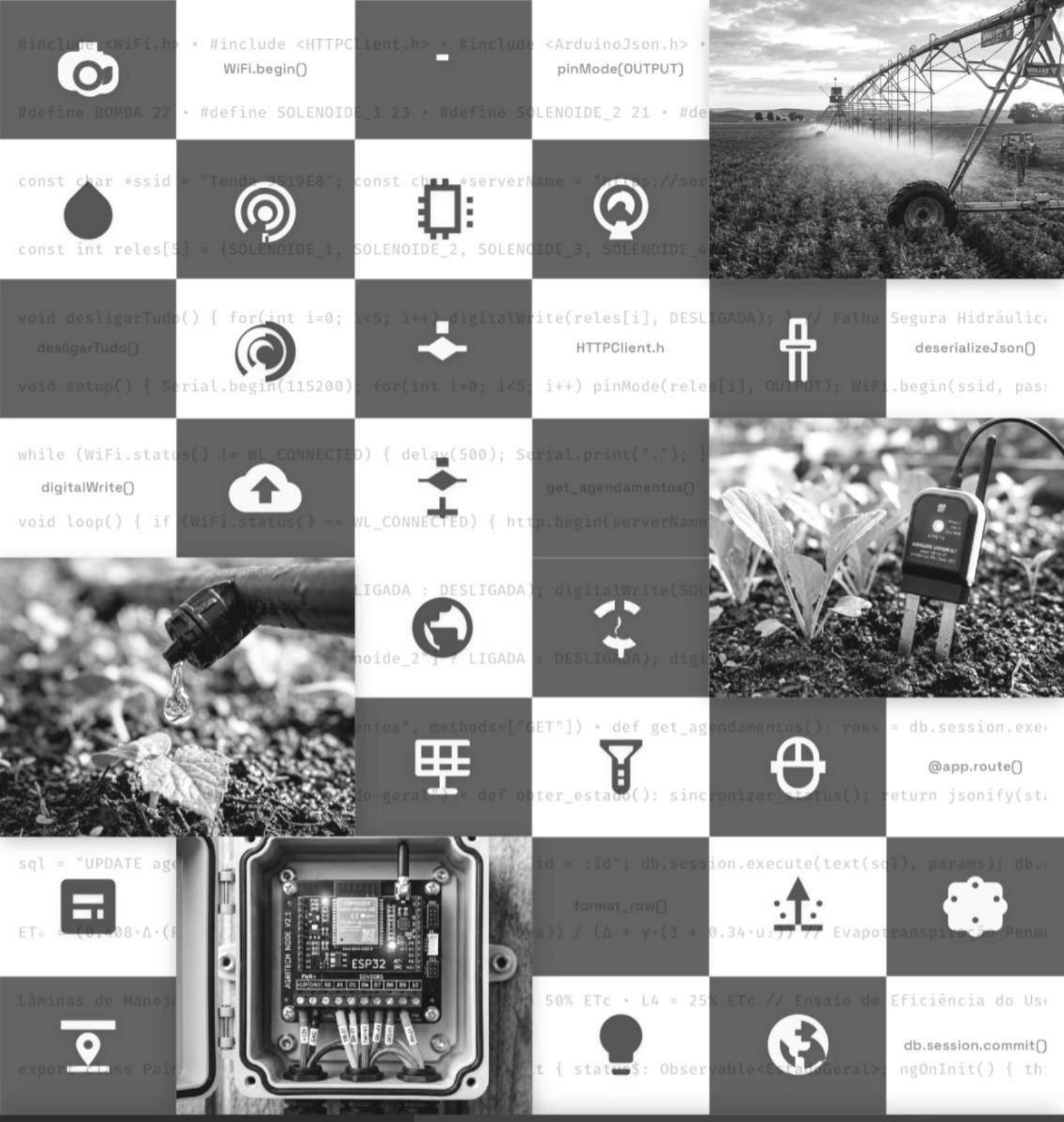
Sergio Souza Novak
Thiago Gratz Spinassé
Adriano Soares de Oliveira Bailão
Marconi Batista Teixeira
Laenor Henrique Tenório Dantas Brandão
Daniel Xavier Vieira
Juliana Fortunato
Leandro Manuel Pereira Sanches
Alexandre Garcia Rezende
Leonardo Nazário Silva dos Santos
Samira Alice de Oliveira
Laila Teixeira de Souza

UM GUIA PARA CONSTRUÇÃO DE

Sistemas Computacionais de Irrigação Automatizados

COM **INTERNET DAS COISAS (IoT)** VOLUME 1

1ª Edição



AUTORES

Sergio Souza Novak
Thiago Gratz Spinassé
Adriano Soares de Oliveira Bailão
Marconi Batista Teixeira
Laenor Henrique Tenório Dantas Brandão
Daniel Xavier Vieira
Juliana Fortunato
Leandro Manuel Pereira Sanches
Alexandre Garcia Rezende
Leonardo Nazário Silva dos Santos
Samira Alice de Oliveira
Laila Teixeira de Souza

UM GUIA PARA CONSTRUÇÃO DE

Sistemas Computacionais de Irrigação Automatizados

COM INTERNET DAS COISAS (IoT) | VOLUME 1

1ª Edição

Editora Omnis Scientia

UM GUIA PARA CONSTRUÇÃO DE SISTEMAS COMPUTACIONAIS DE IRRIGAÇÃO AUTOMATIZADOS COM INTERNET DAS COISAS (IoT)

Volume 1

1ª Edição

RECIFE - PE

2026

EDITOR-CHEFE

Dr. Daniel Luís Viana Cruz

AUTORES

Sergio Souza Novak

Thiago Gratz Spinassé

Adriano Soares de Oliveira Bailão

Marconi Batista Teixeira

Laenor Henrique Tenório Dantas Brandão

Daniel Xavier Vieira

Juliana Fortunato

Leandro Manuel Pereira Sanches

Alexandre Garcia Rezende

Leonardo Nazário Silva dos Santos

Samira Alice de Oliveira

Laila Teixeira de Souza

CONSELHO EDITORIAL

Dr. Amâncio António de Sousa Carvalho - ESS-UTAD - Portugal

Dr. Cássio Brancaleone - UFFS - Brasil

Dr. Marcelo Luiz Bezerra da Silva - UEPa - Brasil

Dra. Pauliana Valéria Machado Galvão - UPE - Brasil

Dr. Plínio Pereira Gomes Júnior - UFRPE - Brasil

Dr. Walter Santos Evangelista Júnior - UFRPE - Brasil

Dr. Wendel José Teles Pontes - UFPE - Brasil

EDITORES DE ÁREA - CIÊNCIAS EXATAS E DA TERRA

Dr. André Luiz Leite Ferreira

Dra. Angela Maria Corrêa Mouzinho Santos

Dr. Filipe Mendonça de Lima

ASSISTENTE EDITORIAL

Thialla Larangeira Amorim

IMAGEM DE CAPA

Canva e OpenAI (ChatGPT)

EDIÇÃO DE ARTE

Nhatallia Laranjeira Amorim

REVISÃO

Os autores

Este trabalho está licenciado com uma Licença Creative Commons – Atribuição-NãoComercial-SemDerivações 4.0 Internacional.

O conteúdo abordado nos artigos, seus dados em sua forma, correção e confiabilidade são de responsabilidade exclusiva dos autores.

Dados Internacionais de Catalogação na Publicação (CIP)
Lumos Assessoria Editorial

G943
Um guia para construção de sistemas computacionais de irrigação automatizados com internet das coisas (IoT) [recurso eletrônico] / Sergio Souza Novak ... [et al.]. — 1. ed. — Recife : Omnis Scientia, 2026.
Dados eletrônicos (pdf).

Inclui bibliografia.
ISBN 978-65-284-0682-1
DOI: 10.47094/978-65-284-0682-1

1. Internet das coisas. 2. Irrigação - Automação. 3. Inteligência artificial - Aplicações agrícolas. 4. Engenharia agrônoma. 5. Abastecimento de água - Gestão - Processamento de dados. 6. Desenvolvimento de recursos hídricos - Processamento de dados. I. Novak, Sergio Souza.

CDD23: 004.678

I-0311261/1

Bibliotecária: Priscila Pena Machado - CRB-7/6971

Editora Omnis Scientia

Av. República do Líbano, nº 251, Sala 2205, Torre A,

Bairro Pina, CEP 51.110-160, Recife-PE.

Telefone: +55 (87) 9920-5762

editoraomnisscientia.com.br

contato@editoraomnisscientia.com.br



AUTORES

Me. Sergio Souza Novak
Mestre em Ciências Agrárias - Agronomia (PPGCA - AGRO) – IF Goiano – Campus Rio Verde - e bacharel em Ciência da Computação pelo Instituto Federal Goiano. Professor na Universidade de Rio Verde (UniRV) especificamente na Faculdade de Engenharia de Software.

Thiago Gratz Spinassé
Aluno de mestrado no Programa de Pós-Graduação em Ciências Agrárias - Agronomia (PPGCA - AGRO) – IF Goiano – Campus Rio Verde. Atua na área de Ciências Agrárias, com ênfase em Irrigação e Drenagem.

Prof. Dr. Adriano Soares de Oliveira Bailão
Docente do curso de Bacharelado em Ciência da Computação do IF Goiano – Campus Rio Verde. Doutor em Ciência da Computação. Atua nas áreas de Ciência da Computação, Inteligência Computacional e Engenharia de Software.

Prof. Pós-Dr. Marconi Batista Teixeira
Coordenador e Docente do Programa de Pós-Graduação em Ciências Agrárias - Agronomia (PPGCA - AGRO) – IF Goiano – Campus Rio Verde.
Atua na área de Irrigação e Drenagem.

Laenor Henrique Tenório Dantas Brandão
Mestrando no Programa de Pós-Graduação em Ciências Agrárias - Agronomia (PPGCA - AGRO) e graduando em Agronomia pelo IF Goiano – Campus Rio Verde. Bacharel em Administração. Atua na área de Ciências Agrárias.

Daniel Xavier Vieira
Aluno de mestrado no Programa de Pós-Graduação em Ciências Agrárias – Agronomia (PPGCA - AGRO) – IF Goiano – Campus Rio Verde. Bacharel em Análise e Desenvolvimento de Sistemas. Atua na área de Ciências Agrárias / Análise e Desenvolvimento de Sistemas.

Dra. Juliana Fortunato
Doutora em Física pela Universidade Estadual de Maringá (UEM), com formação em Bacharelado e Licenciatura em Física pela mesma instituição e Licenciatura em Pedagogia pelo Centro Universitário Unifacvest. Professora da Secretaria de Estado da Educação do Paraná (SEED-PR). Atua no ensino de Física, Matemática e Química na Educação Básica e Profissional Técnica, com experiência em práticas experimentais e Ensino de Ciências.

Me. Leandro Manuel Pereira Sanches
Mestre em Ciências da Computação pela Universidade Federal da Bahia, Salvador, Brasil. Professor na Universidade de Rio Verde (UniRV) na Faculdade de Engenharia de Software. Atua na área de Banco de Dados e Inteligência Artificial (Processamento de Linguagem Natural).

Dr. Alexandre Garcia Rezende
Pós-doutorando no Programa de Pós-Graduação em Ciências Agrárias - Agronomia (PPGCA - AGRO) – IF Goiano – Campus Rio Verde. Doutor em Ciências Agrárias, na área de Ciência do Solo/Física do Solo. Atua na área de Ciências Agrárias.

Dr. Leonardo Nazário Silva dos Santos
Doutor em Engenharia Agrícola (UNICAMP), Mestre em Produção Vegetal e Graduado em Agronomia (UFES). Professor do Instituto Federal do Espírito Santo (IFES), Campus Santa Teresa, e docente vinculado aos programas PPGCA-AGRO e PPGEAS do IF Goiano, Campus Rio Verde. Tem experiência na área de agronomia, com ênfase em recursos hídricos, engenharia de água e solo, uso e manejo de água e uso de água com qualidade inferior.

Samira Alice de Oliveira
Graduanda em Agronomia pelo Instituto Federal do Espírito Santo (IFES) - Campus Santa Teresa. Possui experiência em pesquisa científica com ênfase em olericultura, produção de mudas, manejo de substratos, tecnologia de produção vegetal e aproveitamento de resíduos agroindustriais. Atuou em consultoria e assistência técnica agrícola.

Laila Teixeira de Souza
Engenheira ambiental e de segurança do trabalho. Atua na área de Sustentabilidade.

PREFÁCIO

A transição da agricultura tradicional para a era da Agricultura 4.0 é, acima de tudo, um desafio de integração de diferentes áreas do saber. Embora a tecnologia avance a passos largos, a aplicação de ferramentas digitais no campo frequentemente esbarra em uma lacuna de comunicação: de um lado, estão os agrônomos, que compreendem profundamente as necessidades da terra e da planta; do outro, os profissionais de tecnologia, que dominam a linguagem dos códigos, redes e microcontroladores, mas carecem do contexto biológico.

É exatamente da necessidade de construir uma ponte sólida entre esses dois universos que nasce esta obra. Este livro é o fruto da parceria direta entre dois atores centrais do sistema que você conhecerá ao longo destas páginas: Thiago, um engenheiro agrônomo dedicado a entender os complexos fenômenos de solo, água e planta, e Sergio, um profissional de Tecnologia da Informação especialista em transformar dados físicos em arquiteturas de software responsivas e seguras.

O que apresentamos a você, leitor, não é apenas um manual teórico, mas um convite à prática. Partindo do absoluto zero, unimos a ciência agronômica e a engenharia de computação para guiar a construção completa de um sistema de irrigação automatizado baseado na Internet das Coisas (IoT).

Ao longo dos capítulos, as visões de Thiago e Sergio se complementam continuamente. Enquanto a agronomia dita quando, como e por que irrigar para maximizar a produtividade e economizar os cada vez mais escassos recursos hídricos, a tecnologia da informação responde demonstrando como fazer isso acontecer de forma estável, conectando sensores capacitivos de umidade edáfica a microcontroladores de campo, interligando válvulas solenoides a servidores web, e exibindo todo o ecossistema na palma da mão do produtor por meio de um painel de controle interativo.

Esperamos que a dualidade que permeou a criação deste livro inspire profissionais de ambas as áreas a trabalharem em conjunto. Que os profissionais das ciências agrárias percam o receio de mergulhar nas linhas de código e circuitos eletrônicos, e que os desenvolvedores de software sujem as mãos de terra para entender a realidade do campo.

Boa leitura e excelente trabalho prático!

Thiago Gratz Spinassé e Sergio Souza Novak

SUMÁRIO

CAPÍTULO 1 12
VISÃO GERAL DO SISTEMA, INTERFACES E FUNCIONALIDADES DA PLATAFORMA DE IRRIGAÇÃO IOT
DOI: 10.47094/978-65-284-0682-1/12-22

CAPÍTULO 2 23
EQUIPAMENTOS, SENSORES, ATUADORES E CONTROLADORES NO SISTEMA DE IRRIGAÇÃO AUTOMATIZADO
DOI: 10.47094/978-65-284-0682-1/23-28

CAPÍTULO 3 29
ESQUEMAS ELÉTRICOS, CIRCUITOS DE LEITURA DE SENSORES E ACIONAMENTO DE ATUADORES NO SISTEMA DE IRRIGAÇÃO
DOI: 10.47094/978-65-284-0682-1/29-33

CAPÍTULO 4 34
O PROTOCOLO HTTP E ARQUITETURAS MULTICLIENTE EM SISTEMAS DE IRRIGAÇÃO AUTOMATIZADOS COM INTERNET DAS COISAS (IoT)
DOI: 10.47094/978-65-284-0682-1/34-39

CAPÍTULO 5 40
FIRMWARE C++ PARA ESP32: ARQUITETURA DE CLIENTE MAGRO, SINCRONIZAÇÃO HTTP E ACIONAMENTO DE ATUADORES NA IRRIGAÇÃO IoT
DOI: 10.47094/978-65-284-0682-1/40-55

CAPÍTULO 6 56
O FRONTEND NA AUTOMATIZAÇÃO AGRÍCOLA: ARQUITETURA SPA, TECNOLOGIAS WEB E DECOUPLING DO BACKEND
DOI: 10.47094/978-65-284-0682-1/56-68

CAPÍTULO 7 69
BACKEND PYTHON FLASK: API RESTFUL, ARQUITETURA DE ROTAS, PERSISTÊNCIA MYSQL E TESTES DE INTEGRAÇÃO NA IRRIGAÇÃO IoT
DOI: 10.47094/978-65-284-0682-1/69-91

CAPÍTULO 8 92
CONCLUSÃO E PERSPECTIVAS FUTURAS
DOI: 10.47094/978-65-284-0682-1/92-97

CAPÍTULO 1

VISÃO GERAL DO SISTEMA, INTERFACES E FUNCIONALIDADES DA PLATAFORMA DE IRRIGAÇÃO IOT

1. INTRODUÇÃO E FUNDAMENTAÇÃO DOS SISTEMAS DE IRRIGAÇÃO AUTOMATIZADOS IOT

A modernização da agricultura brasileira pela Incorporação da Internet das Coisas (IoT) e da automação de precisão exige a integração harmoniosa entre duas áreas fundamentais do conhecimento: a agronomia e a tecnologia da informação (TI) (ATZORI; IERA; MO-RABITO, 2010; WOLFERT et al., 2017; BAYIH et al., 2022). Historicamente, a gestão da irrigação no campo operou de forma puramente empírica ou isolada do ecossistema de software. No entanto, o manejo eficiente e sustentável dos recursos hídricos contemporâneos não consiste apenas no acionamento mecânico de motobombas, mas na construção de uma arquitetura computacional completa, capaz de coletar dados ambientais no solo, validar regras de segurança hidráulica, fornece interfaces intuitivas para os operadores e executar rotinas de rega com precisão de segundos.

O objetivo central deste livro é motivar e capacitar tanto **Engenheiros Agrônomos** quanto **Profissionais de TI** a projetar, desenvolver e implantar seus próprios sistemas de irrigação automatizados utilizando a Internet das Coisas. Para os agrônomos, esta obra desmistifica a programação de microcontroladores e o desenvolvimento de software, mostrando como transformar modelos de evapotranspiração e curva de retenção de água no solo em algoritmos executáveis. Para os profissionais de tecnologia da informação, o livro apresenta as demandas reais do ambiente agrícola, ensinando a lidar com restrições de conectividade rural, ruídos eletromagnéticos de motores, calibração de sensores edáficos e arquitetura de borda de campo (*Edge Computing*) (SHI et al., 2016; SATYANARAYA-NAN, 2017; AL-FUQAHA et al., 2015).

Na irrigação tradicional, a gestão da aplicação de água frequentemente enfrenta desafios críticos, tais como a dependência de intervenções manuais propensas a erros humanos, o desperdício de água por sobre irrigação, o risco de sobrecarga e queima de motobombas por operação a seco, a ausência de rastreabilidade histórica dos volumes aplicados e a dificuldade de acompanhar ensaios experimentais agrícolas em tempo real. A introdução de plataformas digitais automatizadas baseadas em IoT soluciona esses gargalos ao integrar a experiência agronômica com a engenharia de software, substituindo equipamentos comerciais fechados de alto custo por soluções abertas, customizáveis e de elevada resiliência.

Este livro foi concebido como um guia prático e didático para motivar essa sinergia interdisciplinar. Ao longo das páginas desta obra, o leitor aprenderá a integrar microcontroladores ESP32 e Arduino com sensores de umidade edáfica, relés e válvulas solenoides; a estruturar um servidor central backend em Python Flask com banco de dados relacional MySQL; a especificar e consumir APIs baseadas no protocolo HTTP REST; e

a construir um painel de controle web moderno e reativo utilizando o framework Angular. Antes de aprofundar nas especificações de código-fonte, protocolos de rede e esquemas eletrônicos de hardware, este primeiro capítulo apresenta a visão geral do sistema que será construído ao longo do livro. São detalhadas todas as telas da plataforma de controle, suas funcionalidades operacionais e o papel de cada módulo na rotina diária de fazendas e estações experimentais agrícolas.

2. PANORAMA DAS FUNCIONALIDADES DO SISTEMA HIPOTÉTICO DE IRRIGAÇÃO

O sistema hipotético de irrigação automatizada projetado neste livro consolida um conjunto abrangente de funcionalidades desenvolvidas para atender tanto aos requisitos de produção agrícola comercial quanto às exigências de pesquisas acadêmicas e experimentação agronômica de campo. A seguir, são descritas em detalhes as principais funcionalidades que compõem a plataforma.

2.1. Autenticação, Controle de Acesso e Gestão de Papéis Interdisciplinar

A plataforma possui um subsistema nativo de controle de acesso para garantir que apenas usuários autorizados interajam com os equipamentos de campo. Os atores principais e operadores do sistema hipotético exemplificados nesta obra são **Thiago Gratz Spinassé**, engenheiro agrônomo, e **Sergio Souza Novak**, profissional de tecnologia da informação, que são autores deste livro. O sistema reflete diretamente essa cooperação interdisciplinar entre as ciências agrárias e a TI, diferenciando os papéis funcionais dos operadores e permitindo definir permissões específicas para profissionais de perfis distintos.

O engenheiro agrônomo Thiago possui privilégios para criar, editar e aprovar rotinas de irrigação, definir lâminas de água com base em dados fenológicos e acompanhar os tratamentos biológicos de campo. Já o especialista em tecnologia da informação Sérgio dispõe de acessos focados na gestão de infraestrutura de rede, servidores backend, monitoramento de endereços IP, gerenciamento de banco de dados e calibração de firmware dos microcontroladores.

A autenticação é realizada através de um formulário de login seguro que valida credenciais de usuário e senha no backend, gerando um token de sessão assinado (JWT) para representar as declarações (*claims*) do usuário de forma íntegra (JONES; BRADLEY; SAKIMURA, 2015; OWASP FOUNDATION, 2024). Esse mecanismo previne acionamentos acidentais ou mal-intencionados que poderiam causar sobre pressão nas tubulações ou estresse hídrico nas plantas (ALGARNI et al., 2023; FERRAG et al., 2020).

2.2. Telemetria e Monitoramento de Atuadores em Tempo Real

O sistema oferece monitoramento contínuo e em tempo real do estado de todos os componentes físicos da rede hidráulica. A interface web comunica-se periodicamente com o servidor central para atualizar o estado operacional da motobomba principal de 5,5

CV (indicando se está LIGADA ou DESLIGADA) e o status de abertura de cada uma das válvulas solenoides instaladas nos setores agrícolas (aberta ou fechada).

Além disso, a plataforma inclui um indicador visual de pulso de rede que sinaliza ativamente a qualidade da conexão entre os microcontroladores de campo (ESP32) e a aplicação central. Se a rede sem fio da fazenda sofrer uma oscilação ou desconexão, o painel alerta imediatamente o usuário sobre a interrupção no fluxo de dados em tempo real.

2.3. Mapeamento Espacial de Experimentos e Tratamentos Agronômicos

Uma das funcionalidades mais inovadoras da plataforma é a capacidade de mapear e exibir graficamente o delineamento de ensaios experimentais agrícolas diretamente no painel de controle. Em estações de pesquisa agronômica, é comum avaliar diferentes tratamentos em função de lâminas de irrigação variadas.

O sistema integra um gráfico cartesiano de dispersão espacial que correlaciona as quatro lâminas de água testadas — L1 (100% da evapotranspiração da cultura), L2 (75%), L3 (50%) e L4 (25%) — com doze posições físicas ao longo da linha de cultivo. Cada parcela experimental é identificada por marcadores visuais codificados por cores, representando os tratamentos aplicados (M1 para microrganismos benéficos, M2, M3 e o grupo controle testemunha). Essa funcionalidade permite ao agrônomo visualizar instantaneamente a distribuição espacial da água e o estado das parcelas experimentais.

2.4. Agendamento Temporal e Filtragem Avançada de Rotinas

A plataforma permite programar rotinas automatizadas de irrigação com precisão de minutos. O usuário pode definir a data de execução, os horários exatos de início e término e a válvula solenoide receptora da água. Todas as rotinas cadastradas são organizadas em uma tabela interativa que permite alternar entre os eventos agendados para o futuro e a guia de histórico de irrigações concluídas.

Para otimizar a gestão em propriedades com múltiplos setores, o sistema disponibiliza um painel de filtros dinâmicos que permite pesquisar agendamentos por código ID, data, intervalo de horário, solenoide/setor específico ou nome do operador responsável. A tabela oferece ainda botões de controle direto para forçar a execução imediata de um agendamento, editar seus parâmetros ou realizar a exclusão preventiva da rotina.

2.5. Validação Automática de Regras Agronômicas e Segurança Hidráulica

Para evitar acionamentos inadequados e proteger a infraestrutura física da fazenda, a plataforma incorpora um mecanismo automático de validação de regras de negócio. Ao cadastrar um novo agendamento por meio de um diálogo modal flutuante, a aplicação calcula dinamicamente a duração da rega em minutos e verifica o atendimento a três salvaguardas fundamentais:

1. Limite de Duração Máxima: A duração contínua de irrigação em um único setor não pode ultrapassar o limite máximo de 30 minutos, evitando o encharcamento

do solo e o desperdício hídrico.

2. Janela de Operação Permitida: O funcionamento dos atuadores é restrito ao horário diurno compreendido entre 07:00 e 19:00, alinhando a irrigação às taxas de evapotranspiração e evitando ruídos de bombas no período noturno.
3. Prevenção de Colisão e Sobrecarga: O sistema impede o agendamento simultâneo de solenoides incompatíveis que excedam a capacidade de vazão da bomba hidráulica ou que causem choque de pressão na rede.
4. Caso qualquer uma dessas regras seja violada, o sistema exibe alertas visuais imediatos no formulário e bloqueia a confirmação do agendamento até que os parâmetros sejam corrigidos.

2.6. Agenda Anual, Linha do Tempo e Estimativa de Volume Hídrico

A gestão agronômica de médio e longo prazo é suportada por um módulo de agenda anual e linha do tempo diária. O sistema apresenta uma matriz de calendário mensal que destaca os dias com irrigações programadas ou realizadas, acumulando o contador de irrigações executadas ao longo do ano.

Ao selecionar um dia específico no calendário, o painel exibe a linha do tempo detalhada dos setores acionados, calculando uma estimativa automatizada do volume total de água em litros (L) aplicado em cada evento com base na vazão nominal dos emissores. Essa informação auxilia o agrônomo no cálculo do balanço hídrico do solo e na otimização dos custos com energia elétrica.

2.7. Sincronização Borda-Nuvem e Resiliência contra Falhas

A plataforma garante a sincronização contínua entre a aplicação web central e os dispositivos de borda de campo (ESP32) por meio de requisições periódicas via protocolo HTTP REST. A cada intervalo ajustável (como 10 segundos), o microcontrolador consulta o backend enviando seu estado e recebendo instruções no formato JSON. Se o servidor indicar que uma rotina está ativa no horário corrente, o ESP32 aciona os relés correspondentes; caso contrário, desativa os circuitos com total segurança.

Se a comunicação de rede for temporariamente perdida, o firmware embarcado mantém o estado seguro desativando as solenoides após um tempo limite (timeout), impedindo que válvulas permaneçam abertas indefinidamente por ausência de sinal.

3. ARQUITETURA MULTICAMADAS: CONCEITUANDO FRONTEND E BACKEND NA IRRIGAÇÃO IOT

Para que engenheiros agrônomos e profissionais de tecnologia da informação consigam projetar e dialogar durante a construção de uma solução de automação agrícola, é fundamental compreender a divisão de papéis da computação em duas grandes camadas fundamentais: o Frontend e o Backend.

3.1. O que é Frontend?

O termo Frontend (também chamado de camada de cliente, camada de apresentação ou interface com o usuário) refere-se a toda a estrutura gráfica, visual e interativa com a qual o usuário humano estabelece contato direto através do navegador de internet no seu computador, tablet ou smartphone.

Desenvolvido utilizando linguagens e tecnologias web como HTML (responsável pela estrutura dos elementos visuais), CSS (responsável pelo estilo, cores e leiaute) e JavaScript/TypeScript (responsável pela dinâmica, reatividade dos componentes e manipulação de eventos), o frontend tem a missão de transformar dados brutos e complexos em informações visuais claras, atraentes e de fácil interpretação.

No contexto do sistema de irrigação proposto neste livro, o frontend é construído com o framework Angular e componentes Angular Material. É no frontend que o agrônomo Thiago ou o profissional de TI Sérgio visualizam os botões de controle das bombas, inspecionam o gráfico espacial dos tratamentos do experimento, preenchem formulários modais de agendamento e consultam a linha do tempo. O frontend captura os cliques e dados digitados pelo operador e os transmite para o servidor, assim como recebe as atualizações da telemetria e as redesenha na tela em tempo real, sem exigir do usuário conhecimentos em comandos de terminal ou linguagens de baixo nível.

3.2. O que é Backend?

O termo Backend (também chamado de camada de servidor, camada de negócios ou motor de processamento) representa a infraestrutura computacional que opera nos bastidores, longe dos olhos do usuário final. Executado em um servidor centralizado (seja uma máquina física instalada na sede da fazenda ou um servidor virtual na nuvem), o backend é o “cérebro” responsável por processar as regras de negócio, gerenciar a persistência de dados em bancos de dados relacionais como o MySQL, impor políticas de segurança e autenticação, e disponibilizar rotas de comunicação através de APIs RESTful.

No sistema de irrigação IoT, o backend desempenha um duplo papel vital. Em primeiro lugar, ele atende às solicitações vindas da aplicação web frontend, autenticando credenciais via tokens de segurança JWT, gravando os agendamentos no banco de dados e verificando se os horários solicitados respeitam as salvaguardas operacionais (como o limite de 30 minutos e a janela entre 07:00 e 19:00). Em segundo lugar, ele atende continuamente aos dispositivos de borda de campo (os microcontroladores ESP32 e Arduino). A cada 10 segundos, os microcontroladores consultam o backend perguntando sobre o estado da rede hidráulica; o backend consulta a tabela de agendamentos no MySQL e responde em formato JSON indicando exatamente quais relés de motobombas e válvulas solenoides devem estar ativos no momento.

Em suma, enquanto o frontend cuida da experiência de uso, da estética e da interação humana, o backend garante a lógica de cálculo, a segurança, o armazenamento de dados e a comunicação automatizada com os equipamentos de campo.

3.3. Roteiro de Aprendizado do Livro Técnico

Para demonstrar a agrônomos e profissionais de TI que é plenamente viável construir um sistema robusto sem a necessidade de adquirir soluções proprietárias inacessíveis, o livro foi estruturado em quatro capítulos complementares que abordam a solução sob a ótica de uma arquitetura em três camadas desacopladas (borda, backend e frontend).

O roteiro de aprendizado do livro foi planejado para guiar ambos os perfis profissionais:

- **Capítulo 1 – Visão Geral do Sistema, Interfaces e Funcionalidades da Plataforma de Irrigação IoT:** Apresenta o panorama conceitual, a motivação para a construção de sistemas próprios, a definição de frontend e backend, a descrição detalhada das funcionalidades e os protótipos de interface gráfica.
- **Capítulo 2 – Equipamentos, Sensores, Atuadores e Controladores no Sistema de Irrigação Automatizado:** Discute a seleção, especificação e montagem de componentes físicos de campo, incluindo microcontroladores ESP32, sensores capacitivos de umidade de solo, módulos relés, contatores trifásicos, bombas centrífugas e válvulas solenoides hidráulicas.
- **Capítulo 3 – O Protocolo HTTP e Arquiteturas Multicliente em Sistemas de Irrigação IoT:** Detalha a especificação das mensagens HTTP, verbos (*GET, POST, PUT, DELETE*), códigos de status, formato JSON e padrões de integração cliente-servidor na Agricultura 4.0.

Nas seções a seguir, cada uma das telas e protótipos da aplicação web é detalhada sob a perspectiva de uso operacional e de design de interface.

4. TELA DE AUTENTICAÇÃO E CONTROLE DE ACESSO (LOGIN)

A tela de autenticação é a primeira interface apresentada ao operador ao acessar a aplicação web. Sua função primária é validar a identidade do usuário antes de permitir o acesso às ferramentas de controle da rede hidráulica.

A Figura 1 apresenta o protótipo gráfico da tela de login e controle de acesso.

Figura 1: Interface gráfica da tela de login e controle de acesso ao sistema de irrigação IoT.



O cartão de autenticação é centralizado na tela sobre um fundo estilizado em gradiente suave. Os campos de entrada para credenciais possuem validações em tempo real e destacam-se ao receber o foco do cursor. A submissão dos dados dispara uma requisição HTTP do tipo POST /login contendo o nome de usuário e a senha criptografada.

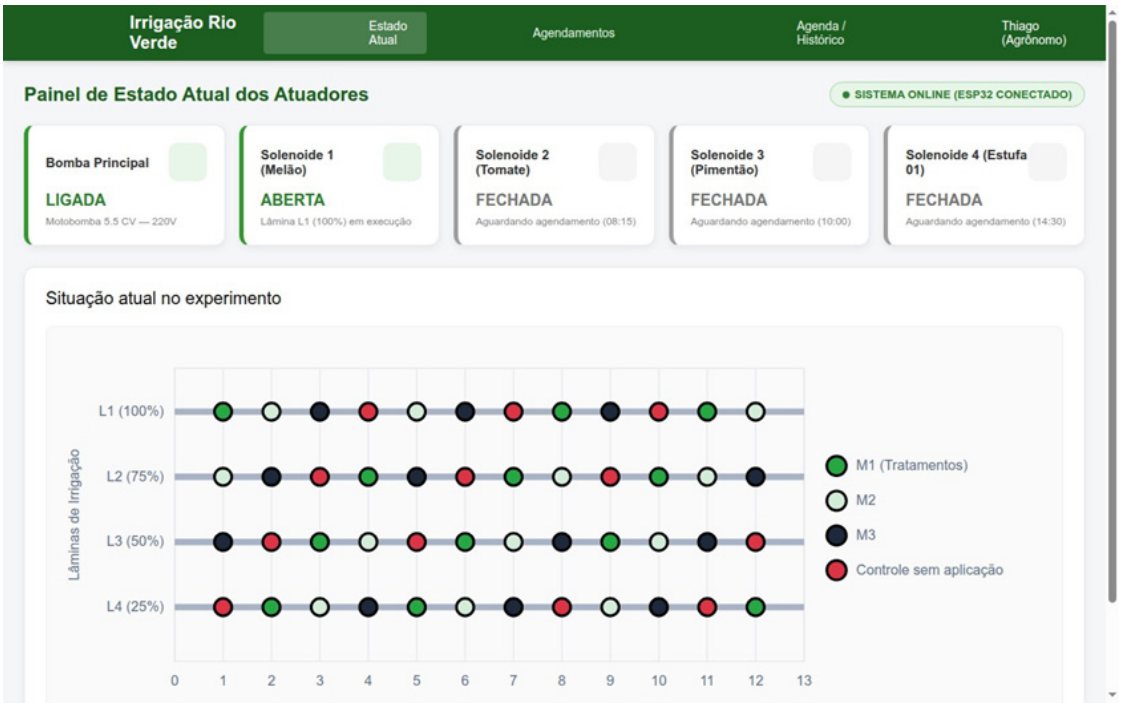
Após a validação no backend Flask, a aplicação recebe um Token Web JSON (JWT) que é anexado a todas as requisições subsequentes. Esse controle permite diferenciar o acesso do engenheiro agrônomo Thiago, que gerencia os parâmetros de rega, do acesso do profissional de TI Sérgio, que monitora a infraestrutura de rede e servidores.

5. PAINEL DE MONITORAMENTO E TELEMETRIA DO EXPERIMENTO (DASHBOARD)

O painel de estado atual reúne as informações essenciais sobre a operação dos equipamentos e o andamento dos ensaios de campo.

A Figura 2 ilustra a interface gráfica do painel de monitoramento e o gráfico do experimento agrícola.

Figura 2: Painel de estado atual dos atuadores hidráulicos e gráfico espacial do experimento agrícola.



No topo da interface, uma barra de status exibe um indicador luminoso contínuo em tom verde que confirma a conectividade em tempo real com o microcontrolador ESP32 de campo. Se houver falha na rede, o indicador altera sua cor para vermelho.

Abaixo do status de conexão, o painel exibe cartões individuais para a motobomba principal de 5,5 CV e para as válvulas solenoides dos setores agrícolas (Solenoide 1 no Melão, Solenoide 2 no Tomate, Solenoide 3 no Pimentão e Solenoide 4 na Estufa 01). Cada cartão destaca o estado do atuador (LIGADA/ABERTA em verde ou DESLIGADA/

FECHADA em cinza).

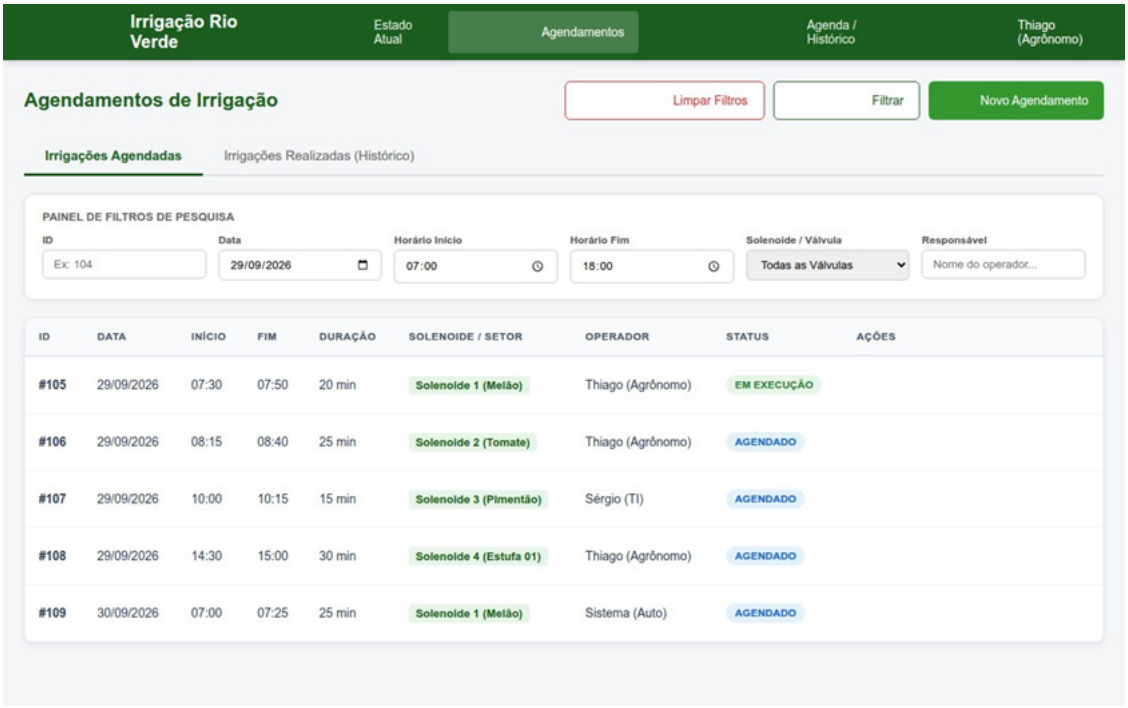
Na parte inferior, o gráfico em matriz de dispersão cartesiana exibe a distribuição espacial do experimento agrícola. O eixo Y representa as quatro lâminas de irrigação avaliadas (L1 a 100%, L2 a 75%, L3 a 50% e L4 a 25%), enquanto o eixo X indica as doze posições espaciais ao longo da linha de cultivo. Pontos coloridos identificam os tratamentos aplicados (M1 para microrganismos benéficos em verde escuro, M2 em verde claro, M3 em preto e Controle testemunha em vermelho).

6. GESTÃO DE AGENDAMENTOS E TABELA DE CONTROLE

A gestão temporal das irrigações é realizada através da tela de agendamentos, que consolida todas as rotinas programadas para os setores da propriedade.

A Figura 3 apresenta a interface da lista de agendamentos com painel de filtros expansível.

Figura 3: Interface de listagem de agendamentos com painel de filtros avançados e ações de controle.



A barra superior contém botões de controle para limpar buscas, alternar o painel de filtros e acionar o cadastro de novos agendamentos. Guias superiores permitem alternar entre as irrigações agendadas e o histórico de irrigações concluídas.

O painel de filtros permite pesquisar agendamentos por código ID, data, intervalo de horário, solenoide ou nome do operador. A tabela central organiza as informações nas

colunas de ID, Data, Início, Fim, Duração, Solenoide/Setor, Operador, Status e Ações. O campo de solenoide utiliza pílulas coloridas para fácil identificação, e o status exibe etiquetas visuais (azul para agendado, verde para em execução e roxo para concluído).

A coluna de ações disponibiliza botões para acionamento manual antecipado da

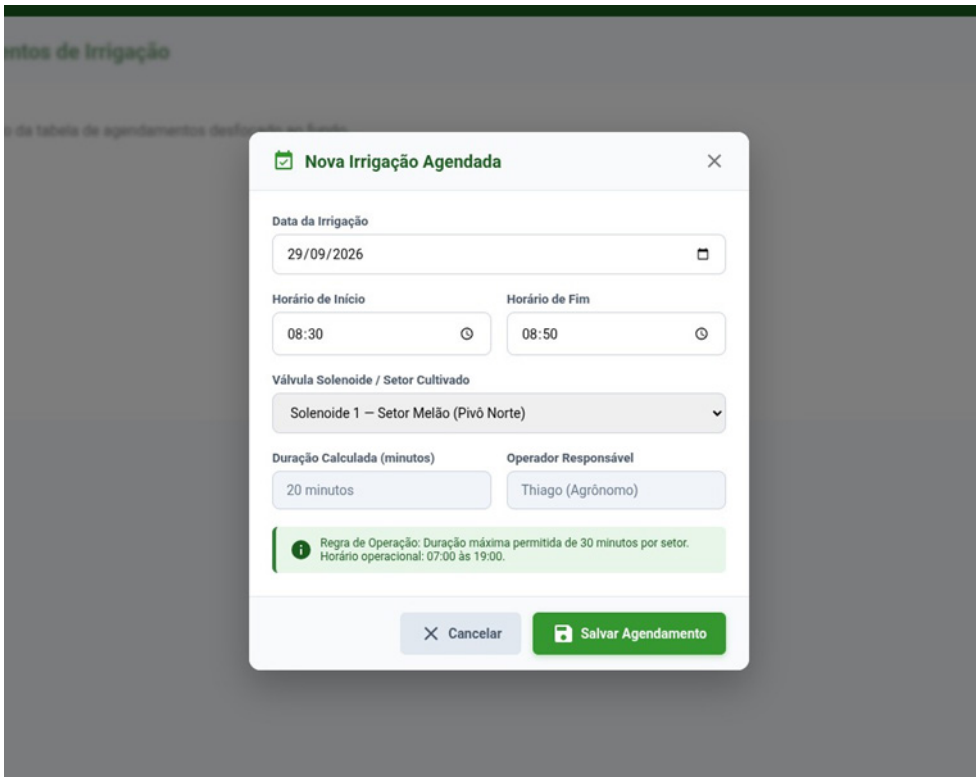
rega, edição de parâmetros ou exclusão preventiva do agendamento.

7. FORMULÁRIO MODAL DE CADASTRO E REGRAS DE VALIDAÇÃO

A inclusão de novas rotinas é realizada através de um diálogo modal flutuante que se sobrepõe à tela de agendamentos, desfocando o conteúdo de fundo.

A Figura 4 ilustra o diálogo modal para cadastramento de novas irrigações.

Figura 4: Diálogo modal para cadastramento de novos agendamentos com validação de regras de segurança.



O operador seleciona a data, os horários de início e término e a solenoide desejada. O formulário calcula automaticamente a duração da rega em minutos e atribui o nome do operador autenticado (como Thiago Agrônomo).

Um painel informativo na parte inferior resume as salvaguardas operacionais: duração máxima de 30 minutos por setor, janela de funcionamento entre 07:00 e 19:00 e bloqueio de colisões entre válvulas. Caso alguma regra seja violada, o botão de salvamento permanece desativado.

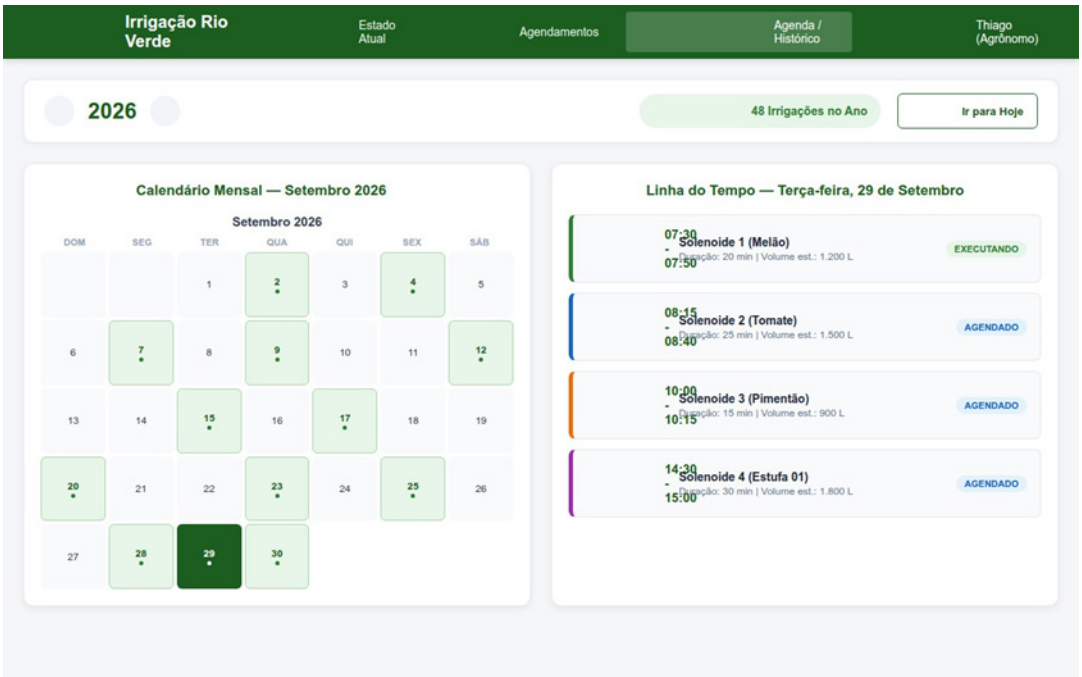
Ao salvar, o frontend dispara uma requisição HTTP do tipo POST/ agendamento para gravar a rotina no banco de dados backend e atualizar a interface de forma reativa.

8. AGENDA ANUAL E LINHA DO TEMPO POR SETOR

O acompanhamento histórico de longo prazo é realizado pela tela de agenda e controle setorial.

A Figura 5 apresenta a visão integrada da agenda anual e da linha do tempo diária.

Figura 5: Visão geral da agenda anual de irrigação e linha do tempo do dia selecionado por setor.



A barra superior disponibiliza navegadores para transição de anos, o contador total de irrigações no ano (48 irrigações em 2026) e um botão de atalho para posicionar o foco no dia atual.

A interface divide-se em dois blocos: o lado esquerdo exibe o calendário mensal com marcadores luminosos nos dias com irrigação programada ou realizada. O lado direito apresenta a linha do tempo do dia selecionado, detalhando os horários, solenoides acionadas, duração e o volume estimado de água em litros que será aplicado pela rede.

Essa visualização permite ao agrônomo correlacionar a lâmina acumulada com a precipitação pluviométrica local, promovendo a gestão eficiente dos recursos hídricos da propriedade.

9. SÍNTESE FUNCIONAL E INTEGRAÇÃO PRÁTICA DA PLATAFORMA

A integração harmoniosa de todas as funcionalidades apresentadas neste capítulo consolida um ambiente computacional robusto e intuitivo para o manejo de irrigação de precisão. O propósito maior desta obra é motivar e capacitar Engenheiros Agrônomos e Profissionais de Tecnologia da Informação a unirem forças, demonstrando que a interseção entre o conhecimento agronômico e a engenharia de software possibilita criar soluções agrícolas inteligentes, autônomas e perfeitamente adaptadas à realidade do campo brasileiro.

Ao compreenderem a separação entre o frontend (camada visual e interativa de telas) e o backend (camada de processamento, banco de dados e controle dos microcontroladores), agrônomos e profissionais de TI adquirem a visão completa de como a Internet das Coisas transforma desafios agronômicos reais em oportunidades de inovação tecnológica.

Nos capítulos a seguir, o leitor será guiado passo a passo na implementação técnica

de cada uma dessas camadas, iniciando pela especificação detalhada do protocolo HTTP e da arquitetura de comunicação cliente-servidor no Capítulo 2.

REFERÊNCIAS

ALGARNI, Abdullah et al. A secure IoT-based irrigation system for precision agriculture. **Sensors**, 2023.

ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. **The Internet of Things: A survey**. [S.l.: s.n.], 2010. v. 54, p. 2787–2805. DOI: 10.1016/j.comnet.2010.05.010.

BAYIH, Tesfaye et al. **Utilization of Internet of Things and Wireless Sensor Networks in Agriculture**. [S.l.]: Springer, 2022.

FERRAG, Mohamed Amine et al. Security and privacy for agricultural IoT: A survey. **IEEE Access**, v. 8, p. 164414–164434, 2020.

AL-FUQAHA, Ala et al. Internet of Things: A survey on enabling technologies, protocols, and applications. **IEEE Communications Surveys and Tutorials**, v. 17, n. 4, p. 2347–2376, 2015. DOI: 10.1109/COMST.2015.2444095.

JONES, Michael B.; BRADLEY, John; SAKIMURA, Nat. **JSON Web Token (JWT)**. [S.l.], 2015. DOI: 10.17487/RFC7519. Disponível em: <https://datatracker.ietf.org/doc/rfc7519/>.

OWASP FOUNDATION. **JSON Web Token Cheat Sheet**. [S.l.: s.n.], 2024. Disponível em: <https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_Cheat_Sheet.ht ml>.

SATYANARAYANAN, Mahadev. The emergence of edge computing. **Computer**, v. 50, n. 1, p. 30–39, 2017. DOI: 10.1109/MC.2017.9.

SHI, Weisong et al. Edge computing: Vision and challenges. **IEEE Internet of Things Journal**, v. 3, n. 5, p. 637–646, 2016. DOI: 10.1109/JIOT.2016.2579198.

WOLFERT, Sjaak et al. Big data in smart farming: A review. **Agricultural Systems**, v. 153, p. 69–80, 2017. DOI: 10.1016/j.agsy.2017.01.023.

CAPÍTULO 2

EQUIPAMENTOS, SENSORES, ATUADORES E CONTROLADORES NO SISTEMA DE IRRIGAÇÃO AUTOMATIZADO

1. INTRODUÇÃO À HARDWARE E INSTRUMENTAÇÃO EM IRRIGAÇÃO IOT

A automação do manejo de irrigação exige uma infraestrutura física composta por equipamentos eletrônicos, hidráulicos e mecânicos capazes de interagir diretamente com o ambiente agrícola. Enquanto os protocolos de rede e os softwares de backend gerenciam o fluxo lógico de informações e as telas de interface fornecem o controle ao operador, são os sensores, controladores e atuadores que traduzem instruções virtuais em ações físicas no campo.

No sistema de irrigação automatizado hipotético apresentado no Capítulo 1, a instrumentação física desempenha um papel central em todas as etapas do ciclo de controle. Um sistema de irrigação inteligente opera em um ciclo contínuo de percepção, processamento e ação. Na etapa de percepção, os sensores capacitivos medem variáveis físicas do solo e a umidade nas parcelas experimentais. Na etapa de processamento, os controladores microprocessados analisam essas medições, sincronizam os dados com o servidor central via requisições HTTP e decidem a abertura das válvulas. Por fim, na etapa de ação, os atuadores elétricos e hidráulicos chaveiam relés de potência, operam a motobomba principal de 5,5 CV e acionam as válvulas solenoides dos setores agrícolas (Melão, Tomate, Pimentão e Estufa 01).

Neste capítulo, são detalhados as principais plataformas de hardware embarcado, os sensores ambientais edáficos, os atuadores elétricos e hidráulicos, as câmeras de inspeção visual e os arranjos experimentais de laboratório utilizados no projeto e na construção do sistema de irrigação automatizado com Internet das Coisas (IoT).

2. CONTROLADORES E PLATAFORMAS EMBARCADAS (EDGE COMPUTING)

O controlador embarcado é o cérebro eletrônico do nó de campo. Ele é responsável por executar o algoritmo de controle de borda, ler os sinais dos sensores locais, acionar os relés e comunicar-se via rede sem fio ou cabeada com o servidor central HTTP em Python Flask.

2.1. Microcontrolador ESP32 e seu Papel no Nó de Campo

O microcontrolador ESP32 consolida-se como a principal plataforma de computação em névoa e borda (Edge Computing) no sistema de irrigação proposto. Equipado com um microprocessador Xtensa Dual-Core de 32 bits operando a frequências de até 240 MHz, o ESP32 integra nativamente conectividade Wi-Fi 802.11 b/g/n e Bluetooth v4.2 BR/EDR e BLE. A Figura 1 ilustra a estrutura física do módulo ESP32 utilizado no nó de controle do sistema.

Figura 6: Módulo microcontrolador ESP32 com conectividade Wi-Fi e Bluetooth integrada para nós de controle de irrigação.



No sistema de irrigação automatizado hipotético, o ESP32 desempenha a função de nó mestre de borda. A cada 10 segundos, seu firmware em C++ dispara uma requisição HTTP do tipo GET /estado-geral para o servidor Flask. O microcontrolador interpreta o documento JSON retornado pelo backend e altera o nível lógico de seus pinos de saída digital GPIO para acionar a bobina da motobomba principal e as bobinas das válvulas solenoides dos quatro setores agrícolas.

O ESP32 possui dois conversores analógico-digitais (ADC) SAR de 12 bits para leitura dos sensores capacitivos de umidade edáfica (ESPRESSIF SYSTEMS, 2024a,b, 2023). Contudo, o fabricante alerta que os canais do ADC2 apresentam restrições de uso quando a conectividade Wi-Fi está ativa. Assim, deve-se especificar cuidadosamente os pinos GPIO do ADC1 para garantir leituras analógicas ininterruptas e confiáveis. O microcontrolador possui pinos com suporte a interrupções externas para medidores de vazão e canais PWM para controle de motores. Sua capacidade de chavear modos de baixo consumo (Deep Sleep) viabiliza o uso de nós autônomos alimentados por pequenas baterias recarregáveis no campo (ATZORI; IERA; MORABITO, 2010; BAYIH et al., 2022).

2.2. Arduino UNO com Ethernet Shield W5100 para Ambientes com Ruído RF

Em cenários rurais onde a infraestrutura sem fio sofre com interferências eletromagnéticas severas provocadas pela partida de motores elétricos trifásicos ou por grandes estruturas metálicas de pivôs centrais, utiliza-se a plataforma Arduino UNO acoplada a um módulo Ethernet cabeado W5100 como alternativa de conexão física resistente a ruídos.

No nó de campo cabeado, o Arduino UNO atua como cliente HTTP via rede local RJ45. Devido às limitações de memória RAM do microcontrolador ATmega328P (2 KB de RAM), a comunicação é otimizada para enviar requisições compactas e decodificar respostas JSON enxutas, garantindo que o acionamento dos relés de potência ocorra com total estabilidade mesmo sob severas interferências elétricas.

3. SENSORES AGRÍCOLAS DE CAMPO E COLETA EDÁFICA

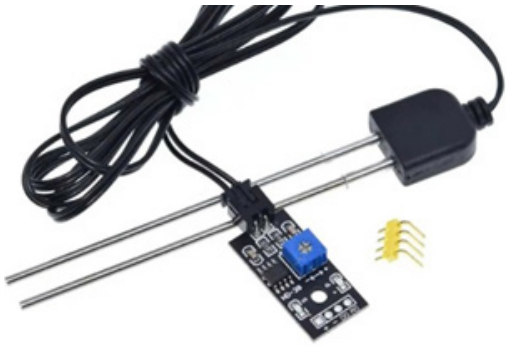
A coleta precisa de dados ambientais é indispensável para evitar tanto o estresse hídrico da cultura por subirrigação quanto o desperdício de recursos e a lixiviação de nutrientes por superirrigação.

3.1. Sensores de Umidade de Solo Capacitivos

Os sensores de umidade do solo são transdutores desenvolvidos para estimar a lâmina de água contida nos poros da matriz edáfica. Diferente dos sensores resistivos convencionais, que sofrem corrosão acelerada por eletrólise e oxidação das hastes metálicas em contato com o solo salino, os sensores de umidade do solo capacitivos operam pela variação da constante dielétrica do meio. A Figura 2 ilustra o sensor capacitivo de umidade do solo empregado nas parcelas agrícolas.

No experimento agrícola exibido no painel de estado atual do sistema (Capítulo 1), esses sensores são posicionados nas doze posições físicas das parcelas de amostragem ao longo das linhas de cultivo. O circuito do sensor capacitivo gera um sinal de alta frequência e mede a alteração na capacitância provocada pela presença de água, já que a constante dielétrica do ar é aproximadamente 1, a do solo seco varia entre 3 e 5, e a da água pura é cerca de 80 (TOPP; DAVIS; ANNAN, 1980; EVETT et al., 2012).

Figura 7: Sensor de umidade do solo capacitivo utilizado para amostragem edáfica em tempo real.



O sinal analógico de saída (variação de tensão) é lido pelos pinos ADC do microcontrolador ESP32. É imperativo ressaltar que a conversão do sinal elétrico em umidade volumétrica exige calibração específica e criteriosa (NAGAHAGE; NAGAHAGE; FU-JINO, 2019; QU et al., 2015). Textura, salinidade, densidade, temperatura e a qualidade do contato sensor-solo influenciam drasticamente a medição (LI et al., 2022; TRACY et al., 2020), demandando validação rigorosa e periódica em condições reais de campo.

4. ATUADORES HIDRÁULICOS E DISPOSITIVOS DE CHAVEAMENTO ELÉTRICO

Os atuadores são os elementos finais de controle que convertem os sinais de comandos emitidos pelos microcontroladores em trabalho físico, promovendo o deslocamento de fluídos e o acionamento de potências elétricas elevadas.

4.1. Válvulas Solenoides Hidráulicas por Setor

A válvula solenoide é um atuador eletromecânico utilizado para interromper ou liberar a passagem de água nas tubulações dos setores de irrigação. Ela é composta por uma bobina elétrica (solenoide) e um corpo de válvula com diafragma interno. A Figura 3 ilustra uma válvula solenoide hidráulica para uso agrícola.

No sistema de irrigação automatizado, quatro válvulas solenoides são instaladas na tubulação principal de recalque, correspondendo à Solenoide 1 (Setor de Melão), Solenoide 2 (Setor de Tomate), Solenoide 3 (Setor de Pimentão) e Solenoide 4 (Estufa 01). Quando a bobina da solenoide é energizada pelo módulo relé do ESP32, um êmbolo metálico interno abre o orifício de alívio de pressão. O fechamento e a abertura hidraulicamente controlados da válvula amortecem mudanças abruptas de fluxo, evitando o golpe de aríete na rede hidráulica – um pico transitório de pressão (CHAUDHRY, 2014; WY-LIE; STREETER; SUO, 1993). Evitar a operação contínua da motobomba principal contra válvulas fechadas é igualmente vital, a fim de prevenir danos severos ao conjunto motobomba por cavitação ou operação em fluxo zero (KARASSIK et al., 2008; STEPHENSON, 1989).

Figura 8: Válvula solenoide de controle hidráulico para direcionamento do fluxo de água nos setores de plantio.



4.2. Motobomba Hidráulica Principal de Pressurização

A bomba hidráulica é o equipamento de pressurização responsável por captar a água do reservatório ou poço artesiano e recalcá-la até os emissores de irrigação (gotejadores ou aspersores). A Figura 4 apresenta a bomba hidráulica centrífuga acoplada ao sistema.

A motobomba principal do sistema hipotético possui potência de 5,5 CV alimentada em tensão trifásica de 220V. Sua operação é integrada logicamente às válvulas solenoides: a bomba somente é ativada pelo backend e pelo controlador ESP32 quando ao menos uma das quatro válvulas solenoides estiver aberta, evitando a pressurização com tubulação

fechada (operação contra válvula fechada), o que causaria sobreaquecimento do rotor e queima do motor elétrico.

Figura 9: Bomba hidráulica centrífuga de 5,5 CV acoplada ao sistema de pressurização de água.



4.3. Módulos Relé e Contatores Magnéticos de Potência

Para que os sinais de controle de baixa tensão (3.3V DC emitidos pelo ESP32) consigam acionar equipamentos de alta potência (solenoides de 24V e motobombas de 220V/380V), é indispensável utilizar elementos de isolamento galvânico e chaveamento de potência. A Figura 5 ilustra o módulo relé optoacoplado e a contatora magnética industrial empregados no painel de comando elétrico.

Figura 10: Módulo relé optoacoplado (esquerda) e contatora magnética de potência (direita) para isolamento elétrico e acionamento seguro dos motores.





O módulo relé optoacoplado utiliza um LED interno e um fototransistor para isolar totalmente o circuito lógico do microcontrolador ESP32 das tensões dos atuadores. Os contatos secos do relé chaveiam a bobina de comando da contatora magnética industrial. A contatora, por sua vez, fecha os contatos trifásicos de alta corrente que alimentam o motor elétrico da motobomba de 5,5 CV, garantindo a proteção contra sobrecarga por meio de relés térmicos acoplados.

REFERÊNCIAS

ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. **The Internet of Things**: A survey. [S.l.: s.n.], 2010. v. 54, p. 2787–2805. DOI: 10.1016/j.comnet.2010.05.010.

BAYIH, Tesfaye et al. **Utilization of Internet of Things and Wireless Sensor Networks in Agriculture**. [S.l.]: Springer, 2022.

CHAUDHRY, M. Hanif. **Applied Hydraulic Transients**. 3. ed. New York: Springer, 2014. DOI: 10.1007/978-1-4614-8538-4.

ESPRESSIF SYSTEMS. **ESP32 Series Datasheet**. [S.l.], 2024. Disponível em: <https://documentation.espressif.com/esp32_datasheet_en.html>.

ESPRESSIF SYSTEMS. **ESP32 Technical Reference Manual**. [S.l.], 2024. Disponível em: <https://documentation.espressif.com/esp32_technical_reference_manual_en.html>.

ESPRESSIF SYSTEMS. **ESP32-WROOM-32 Datasheet**. [S.l.], 2023. Disponível em:<https://documentation.espressif.com/esp32-wroom-32_datasheet_en.pdf>.

EVETT, Steven R. et al. Soil water sensing for water balance, ET and WUE. **Agricultural Water Management**, v. 104, p. 1–9, 2012. DOI: 10.1016/j.agwat.2011.12.002.

KARASSIK, Igor J. et al. **Pump Handbook**. 4. ed. New York: McGraw-Hill, 2008.

LI, X. et al. Soil moisture measuring techniques and factors affecting the moisture dynamics: A comprehensive review. **Sustainability**, v. 14, n. 18, p. 11538, 2022. DOI: 10.3390/su141811538.

NAGAHAGE, E. A. A. D.; NAGAHAGE, I. S. P.; FUJINO, T. Calibration and validation of a low-cost capacitive moisture sensor to integrate the automated soil moisture monitoring system. **Agriculture**, v. 9, n. 7, p. 141, 2019. DOI: 10.3390/agriculture9070141.

QU, W. et al. Calibration of a novel low-cost soil water content sensor. **Sensors**, v. 15, n. 9, p. 24127–24143, 2015. DOI: 10.3390/s150924127.

STEPHENSON, D. **Pipeline Design for Water Engineers**. Amsterdam: Elsevier, 1989.

TOPP, G. C.; DAVIS, J. L.; ANNAN, A. P. Electromagnetic determination of soil water content: Measurements in coaxial transmission lines. **Water Resources Research**, v. 16, n. 3, p. 574–582, 1980. DOI: 10.1029/WR016i003p00574.

TRACY, S. R. et al. Review of novel and emerging proximal soil moisture sensors for use in agriculture. **Sensors**, v. 20, n. 23, p. 6934, 2020. DOI: 10.3390/s20236934.

WYLIE, E. Benjamin; STREETER, Victor L.; SUO, Lisheng. **Fluid Transients in Systems**. Englewood Cliffs: Prentice Hall, 1993.

CAPÍTULO 3

ESQUEMAS ELÉTRICOS, CIRCUITOS DE LEITURA DE SENSORES E ACIONAMENTO DE ATUADORES NO SISTEMA DE IRRIGAÇÃO

1. INTRODUÇÃO À INSTRUMENTAÇÃO ELÉTRICA E CIRCUITOS DE CAMPO

A confiabilidade de um sistema automatizado de irrigação depende criticamente da qualidade dos circuitos eletrônicos projetados para realizar a interface entre os dispositivos de campo e o microcontrolador embarcado. No Sistema de Irrigação e Leitura de Campo (SILF), o hardware precisa executar duas tarefas fundamentais e complementares: a leitura contínua dos dados edafoclimáticos (umidade do solo, umidade relativa do ar e temperatura ambiente) e o acionamento seguro dos atuadores hidráulicos de alta potência (motobomba centrífuga e válvulas solenoides).

A concepção dos circuitos eletrônicos exige atenção rigorosa às características elétricas dos sensores e atuadores, bem como aos níveis de tensão suportados pelos microcontroladores. Enquanto o ESP32 opera nativamente com lógica de 3,3V em suas portas de entrada e saída (GPIO), as cargas de campo frequentemente demandam tensões de 12V, 24V ou até 220V em corrente alternada. Portanto, o projeto dos circuitos de leitura e acionamento deve garantir o condicionamento de sinal adequado, a filtragem de ruídos eletromagnéticos e o isolamento elétrico necessário para proteger os componentes lógicos contra sobretensões e picos de corrente.

Neste capítulo, são apresentados os esquemas elétricos completos, a especificação dos sensores edafoclimáticos, o circuito de leitura de dados, a lógica de filtragem e média computacional embarcada, e o circuito de acionamento de potência utilizados na plataforma de irrigação automatizada.

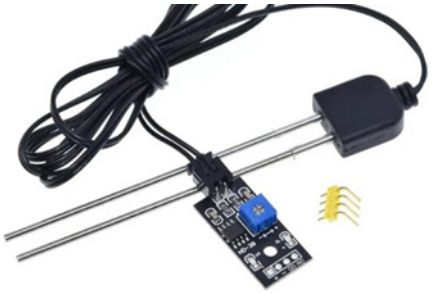
2. ESPECIFICAÇÃO DOS SENSORES EDAFOCLIMÁTICOS

A caracterização precisa do microclima e da matriz edáfica exige a seleção de sensores com adequada sensibilidade, baixo consumo energético e elevada durabilidade quando submetidos às intempéries do ambiente agrícola.

2.1. Sensor de Temperatura e Umidade Relativa do Ar DHT11

Para o monitoramento das condições atmosféricas que influenciam diretamente as taxas de evapotranspiração das culturas, o sistema utiliza o sensor integrado do tipo DHT11. A escolha do sensor DHT11 deve-se ao fato de ser uma solução amplamente consagrada no monitoramento de temperatura e umidade, destacando-se pela simplicidade de operação, confiabilidade e baixo custo de aquisição. A Figura 1 ilustra os sensores utilizados no trabalho.

Figura 11: Sensores edafoclimáticos utilizados no sistema de irrigação automatizado.



O sensor de temperatura e umidade do ar DHT11 possui uma faixa de medição compreendida entre 0 °C e 50 °C para temperatura ambiente e de 20% a 80% para a umidade relativa do ar, apresentando precisão nominal de ±2 °C e ±5%, respectivamente. O componente integra um elemento capacitivo para mensuração da umidade e um termistor NTC para medição da temperatura, convertendo os sinais analógicos internos em um protocolo digital proprietário de barramento único (1-Wire).

Para assegurar a estabilidade do sinal digital e evitar flutuações de tensão na linha de comunicação durante as leituras pelo microcontrolador ESP32, é empregado um resistor de elevação (pull-up) conectado entre o pino de dados do sensor e a linha de alimentação de 3,3V.

2.2. Sensores Capacitivos de Umidade de Solo Anticorrosivos

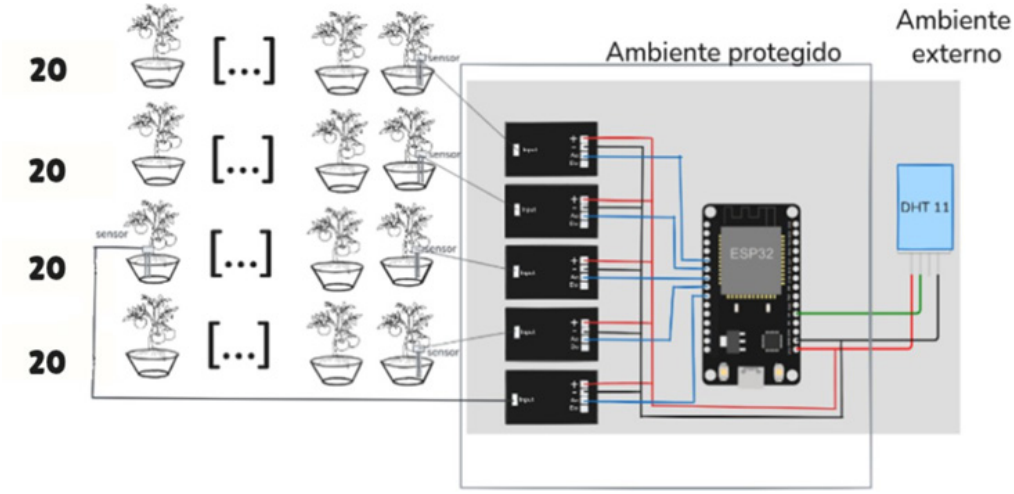
O monitoramento da lâmina de água retida no solo é realizado por cinco sensores de umidade do solo do tipo anticorrosivo, desenvolvidos pela fabricante WJ Componentes Eletrônicos. A característica anticorrosiva é fundamental para a operação contínua em ambiente edáfico, uma vez que previne a degradação e a oxidação prematura dos eletrodos provocadas pelo contato direto com a umidade e com os sais presentes na solução do solo.

Ao todo, foram utilizados cinco sensores desse modelo distribuídos estrategicamente pelas parcelas de amostragem. Cada sensor é acompanhado por um módulo eletrônico de condicionamento que converte a variação de capacitância captada pelas hastes em dois tipos de saída: uma saída analógica (tensão proporcional à umidade) e uma saída digital (nível lógico ajustado por um trimpot comparador). O módulo adapta os níveis de saída perfeitamente às faixas de entrada de 0 a 3,3V aceitas pelos conversores analógico-digitais (ADC) do microcontrolador ESP32.

3. CIRCUITO DE LEITURA DE DADOS E INFRAESTRUTURA DE SINAL

No Sistema de Irrigação e Leitura no Campo, a comunicação entre as placas microcontroladas de leitura e os circuitos de acionamento ocorre de forma integrada e contínua. A Figura 2 apresenta o esquema detalhado do circuito de leitura dos dados edafoclimáticos.

Figura 12: Circuito de ligação dos cinco sensores de umidade do solo e do sensor DHT11 ao microcontrolador ESP32.



O esquema elétrico do circuito de leitura demonstra a conexão direta das saídas analógicas dos cinco sensores de umidade do solo aos pinos de conversão analógico-digital (ADC) do ESP32, bem como a ligação do pino de dados do sensor DHT11 com seu respectivo resistor de elevação (pull-up).

Uma vantagem relevante desse projeto de circuito reside na compatibilidade direta de tensão: tanto os sensores capacitivos de umidade do solo quanto o sensor DHT11 operam na tensão nativa de 3,3V fornecida pelo próprio microcontrolador ESP32. Com isso, elimina-se a necessidade de transformadores, divisores de tensão ou conversores de nível lógico adicionais na etapa de leitura, reduzindo a complexidade do circuito e aumentando a confiabilidade do hardware.

4. TRATAMENTO COMPUTACIONAL, FILTRAGEM E MÉDIA DOS SENSORES

A amostragem de dados em ambientes agrícolas está sujeita a interferências causadas por desconexão física de cabos, mau contato, sensores danificados ou ruídos induzidos. Para garantir a obtenção de valores de umidade de solo altamente confiáveis, o firmware do ESP32 executa rotinas de validação e tratamento antes do envio das medições ao servidor central.

Após a realização das leituras pelos pinos ADC, o código de programação realiza uma verificação de consistência para checar se cada sensor está realmente conectado e coletando dados de forma correta (verificando se o sinal de tensão encontra-se dentro dos limites operacionais de calibração). Caso um sensor apresente circuito aberto ou leitura fora da faixa válida, ele é automaticamente desconsiderado do cálculo.

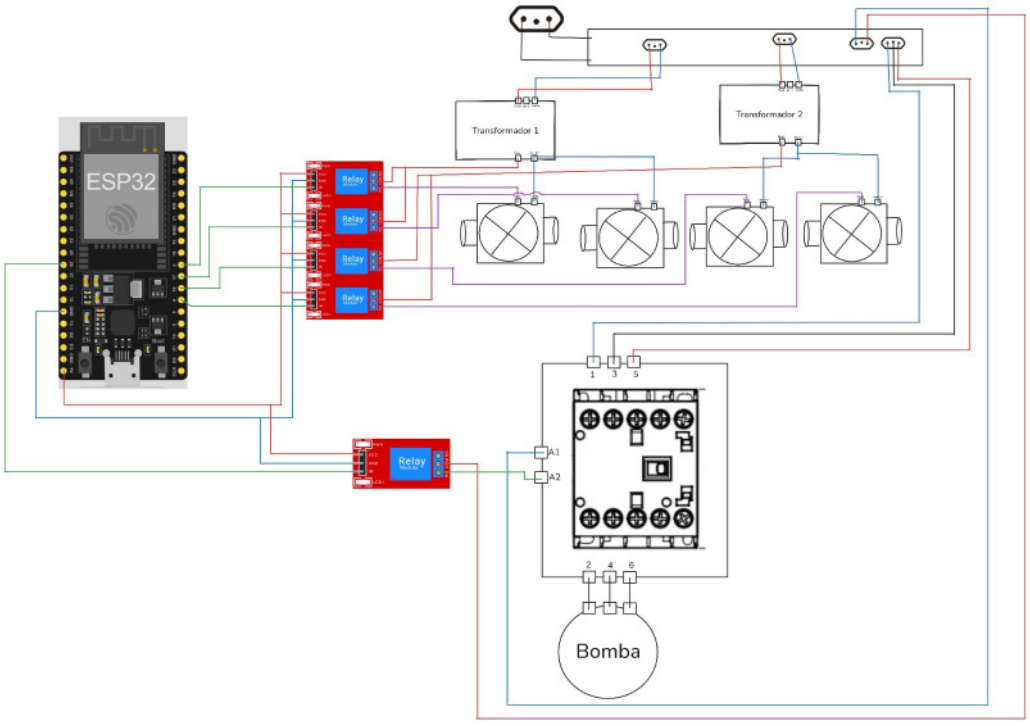
Em seguida, o algoritmo calcula a média aritmética considerando exclusivamente os valores provenientes dos sensores ativos. No entanto, em aplicações críticas, a simples média aritmética não é suficiente para isolar falhas intrínsecas, valores discrepantes (outliers) ou o deslocamento de sensores. Recomenda-se o uso suplementar da mediana, média aparada, filtros preditivos de Kalman e técnicas robustas de validação temporal

(OPPENHEIM; SCHAFER, 1999; SMITH, 1997; WELCH; BISHOP, 1995; HAMPEL et al., 1986). Por fim, a variável de controle processada e os valores brutos individuais são empacotados em um documento JSON (BRAY, 2017) e transmitidos via requisição HTTP POST para a API em Python Flask (GRINBERG, 2018), a qual persiste os dados de telemetria no banco relacional MySQL para monitoramento visual no painel web (FEW, 2013; SHNEIDERMAN et al., 2016).

5. CIRCUITO DE ACIONAMENTO DE ATUADORES DE POTÊNCIA

Enquanto o circuito de leitura cuida da captação de dados de entrada, o circuito de acionamento é responsável por executar o comando final de acionamento das cargas hidráulicas e elétricas de alta potência. A Figura 3 ilustra o diagrama esquemático do circuito de acionamento do sistema.

Figura 13: Circuito de acionamento dos relés de potência, motobomba principal e válvulas solenoides.



O circuito de acionamento utiliza os pinos de saída digital GPIO do ESP32 para comandar os módulos relé de potência. Cada canal do módulo relé possui um optoacoplador interno que isola galvanicamente o microcontrolador da rede de potência dos atuadores.

Quando o backend Flask indica que um setor deve ser irrigado, a GPIO correspondente do ESP32 é levada a nível lógico alto (3,3V), energizando o LED do optoacoplador e disparando o transistor de acionamento do relé. O contato seco do relé fecha o circuito de comando de 220V que alimenta a bobina da contatora magnética industrial. A contatora fecha seus contatos trifásicos de alta corrente, ligando a motobomba principal de 5,5 CV. Simultaneamente, outro canal do relé energiza a bobina da válvula solenoide do setor selecionado (Melão, Tomate, Pimentão ou Estufa), permitindo a passagem controlada da

água.

Para proteger os transistores e relés contra surtos de tensão provocados pela desenergização de cargas indutivas (como as bobinas das solenoides e contatores), o circuito integra diodos de roda livre (flyback diodes) em paralelo com os enrolamentos indutivos, dissipando a energia armazenada e garantindo longa vida útil ao painel de comando.

Com os circuitos de leitura e acionamento totalmente detalhados, o próximo capítulo aborda a especificação e o funcionamento do protocolo HTTP REST utilizado para integrar esse hardware à aplicação web central.

REFERÊNCIAS

BRAY, Tim. **The JavaScript Object Notation (JSON) Data Interchange Format**. [S.l.], 2017. DOI: 10.17487/RFC8259. Disponível em: <https://www.rfc-editor.org/rfc/rfc8259.html>.

FEW, Stephen. **Information Dashboard Design: Displaying Data for At-a-Glance Monitoring**. 2. ed. Burlingame: Analytics Press, 2013.

GRINBERG, Miguel. **Flask Web Development: Developing Web Applications with Python**. 2. ed. Sebastopol: O'Reilly Media, 2018.

HAMPEL, Frank R. et al. **Robust Statistics: The Approach Based on Influence Functions**. New York: John Wiley e Sons, 1986.

OPPENHEIM, Alan V.; SCHAFER, Ronald W. **Discrete-Time Signal Processing**. 2. ed. Upper Saddle River: Prentice Hall, 1999.

SHNEIDERMAN, Ben et al. **Designing the User Interface: Strategies for Effective Human-Computer Interaction**. 6. ed. Boston: Pearson, 2016.

SMITH, Steven W. **The Scientist and Engineer's Guide to Digital Signal Processing**. San Diego: California Technical Publishing, 1997.

WELCH, Greg; BISHOP, Gary. **An Introduction to the Kalman Filter**. [S.l.: s.n.], 1995.

CAPÍTULO 4

O PROTOCOLO HTTP E ARQUITETURAS MULTICLIENTE EM SISTEMAS DE IRRIGAÇÃO AUTOMATIZADOS COM INTERNET DAS COISAS (IoT)

1. INTRODUÇÃO AO PROTOCOLO HTTP NA INTERNET DAS COISAS

A transformação digital no setor agrícola, impulsionada pelo conceito de Agricultura 4.0, exige a integração contínua entre dispositivos de campo (atuadores, relés e sensores edáficos) e sistemas de gestão de alto nível (dashboards web e aplicativos móveis). No centro dessa infraestrutura de comunicação encontra-se o protocolo HTTP (*Hypertext Transfer Protocol*), um protocolo da camada de aplicação do modelo OSI/TCP-IP que se consagrou como o padrão fundamental para a transferência de dados na Web e em arquiteturas distribuídas de Internet das Coisas (IoT).

O funcionamento do HTTP baseia-se no modelo de requisição e resposta, no qual o cliente solicita dados ou instrui ações e o servidor retorna as respostas correspondentes. O HTTP é caracterizado como um protocolo sem estado (stateless), uma vez que o servidor não mantém, por padrão, informações sobre requisições anteriores efetuadas pelo mesmo cliente. Dessa forma, cada requisição é tratada de maneira estritamente independente das solicitações prévias.

No contexto de sistemas de irrigação automatizados, essa propriedade sem estado proporciona elevada escalabilidade e simplicidade operacional para dispositivos de recursos limitados, como o ESP32 e o Arduino. Cada consulta periódica de estado (polling) ou acionamento pontual de bomba hidráulica é executado como uma transação autônoma perante o servidor central Flask.

Para um engenheiro agrônomo ou profissional de TI, compreender o protocolo HTTP torna-se intuitivo ao compará-lo à dinâmica de ordens de serviço e relatórios de manejo entre a administração de uma propriedade agrícola e suas equipes de campo. Da mesma forma que um agrônomo envia uma solicitação formal indicando qual talhão deve ser inspecionado e aguarda o retorno com os indicadores fitossanitários ou térmicos, uma requisição HTTP representa uma mensagem padronizada emitida por um dispositivo no campo solicitando dados ou instruindo uma ação específica a um servidor central de dados.

Na prática da Agricultura 4.0 e da irrigação de precisão, a tomada de decisão agronômica — como a determinação do momento ideal de rega, o controle da lâmina de água ou o monitoramento do estresse hídrico da cultura — depende do fluxo contínuo e confiável de informações colhidas no solo (STAFFORD, 2000; PIERCE; NOWAK, 1999). O protocolo HTTP funciona como o veículo universal de dados da lavoura digital: ele empacota as medições coletadas pelos sensores em mensagens estruturadas e as transporta via rede de comunicação até a plataforma de gestão, assegurando que o sistema interprete as variáveis agronômicas sem ruídos ou ambiguidades (FIELDING; NOTTINGHAM; RESCHKE, 2022; BERNERS-LEE; FIELDING; FRYSTYK, 1996).

2. A ARQUITETURA CLIENTE-SERVIDOR EM SISTEMAS DE IRRIGAÇÃO IOT

A organização computacional de uma solução de automação agrícola baseia-se na separação clara de responsabilidades entre duas entidades fundamentais: os clientes de amostragem/interface e o servidor central de processamento e persistência. Essa abordagem, conhecida como Arquitetura Cliente-Servidor, é a espinha dorsal que permite interconectar dispositivos com capacidades computacionais completamente distintas.

Na arquitetura cliente-servidor aplicada à irrigação inteligente, os clientes ocupam a camada inferior do sistema. Eles são os agentes que interagem diretamente com o ambiente físico (coletando medições de umidade através de sensores no solo ou acionando relés em motores) e com os operadores humanos (renderizando dashboards web e capturando comandos). O servidor, acompanhado do banco de dados relacional MySQL, ocupa a camada superior da arquitetura, atuando como o repositório central de verdade, o motor de regras de negócio e a autoridade única que válida e autoriza a execução de irrigações.

A Figura 1 apresenta o diagrama esquemático da arquitetura cliente-servidor implementada no sistema de irrigação IoT, destacando os clientes dispostos na camada inferior e o servidor central com o banco de dados posicionado na camada superior.

A análise detalhada do diagrama da Figura 1 revela o fluxo de comunicação e o papel desempenhado por cada camada na automação agrícola:

Figura 14: Diagrama da arquitetura cliente-servidor em irrigação IoT: os clientes multimodais na camada inferior emitem requisições HTTP para a camada superior central de servidor e banco de dados.



2.1. O Papel da Camada Inferior: Clientes Multimodais de Campo e Interface

Os clientes estão posicionados na camada inferior do sistema porque são os pontos finais de captura de dados e de apresentação visual. Eles dividem-se em dois tipos principais de dispositivos:

1. **Clientes de Interface Humana (Frontend Web SPA):** Executado nos navegadores de computadores de mesa ou dispositivos móveis utilizadores por Thiago (Agrônomo) e Sérgio (TI), o cliente web Angular envia requisições HTTP do tipo POST/login para autenticação e POST /agendamentos para cadastrar rotinas de rega. Ele também consome dados via GET para renderizar os cartões de status e os gráficos espaciais de tratamento.
2. **Clientes Embarcados de Campo (ESP32 e Arduino):** Instalados nas parcelas agrícolas, esses nós microcontrolados atuam como clientes autônomos. A cada 10 segundos, o ESP32 emite uma requisição do tipo GET / estado-geral em busca de instruções de acionamento de relés. Já o nó de leitura com Arduino transmite amostras dos sensores capacitivos de umidade edáfica via POST /telemetria.

É fundamental destacar que os clientes nunca se comunicam diretamente entre si. Um nó de campo ESP32 não troca mensagens diretamente com o navegador web do agrônomo. Toda a interatividade intermediada ocorre obrigatoriamente através da camada superior.

2.2. O Papel da Camada Superior: Servidor Central Backend e Banco de Dados

O servidor backend em Python Flask e o banco de dados relacional MySQL ocupam a camada superior da arquitetura porque concentram a autoridade decisória e a inteligência do sistema. Suas atribuições incluem:

1. **Processamento e Validação de Regras de Negócio:** Ao receber uma requisição de agendamento vinda do cliente web, o servidor Flask revalida as salvaguardas de segurança (verificando se a duração ultrapassa 30 minutos ou se há colisão de horários entre solenoides).
2. **Persistência e Fonte Única de Verdade:** O banco MySQL armazena o histórico de telemetrias, os agendamentos cadastrados e a tabela de permissões de usuários. Isso garante que, mesmo que o navegador web seja fechado ou que um nó de campo seja reiniciado por oscilação de energia, os dados permaneçam preservados com total integridade.
3. **Coordenação de Ações e Respostas JSON:** O servidor processa a consulta do ESP32, verifica a hora atual no banco de dados e retorna a resposta HTTP 200 OK contendo o documento JSON que instrui a abertura ou fechamento das solenoides.

Essa hierarquia garante um sistema extremamente resiliente, desacoplado e imune à perda de dados local nos nós de borda.

3. MÉTODOS E VERBOS HTTP NA AUTOMAÇÃO AGRÍCOLA

As requisições HTTP emitidas pelos clientes da camada inferior utilizam métodos

específicos (conhecidos como verbos HTTP) para indicar a intenção da operação solicitada ao servidor backend na camada superior. Cada método possui semântica própria alinhada aos padrões de arquitetura REST (Representational State Transfer).

3.1. Método GET: Leitura e Telemetria

O método GET é utilizado exclusivamente para consultar e recuperar informações do servidor, sem provocar alterações de estado no banco de dados. Na plataforma de irrigação, o método GET é empregado em duas situações principais:

1. **Consulta de Estado pelos Microcontroladores:** O ESP32 realiza requisições do tipo GET /estado-geral a cada 10 segundos para verificar se a motobomba principal e as válvulas solenoides dos quatro setores (Melão, Tomate, Pimentão e Estufa 01) devem estar ativas ou inativas.
2. **Carregamento de Dados pelo Frontend:** A aplicação Angular utiliza o método GET para buscar a lista de agendamentos futuros, recuperar o histórico de acionamentos e carregar os pontos do experimento agrícola exibidos no gráfico do painel de controle.

Por ser um método seguro e idempotente, múltiplas requisições GET idênticas retornam o mesmo resultado sem gerar efeitos colaterais na infraestrutura física de campo.

3.2. Método POST: Criação e Envio de Comandos

O método POST é empregado para submeter dados ao servidor, promovendo a criação de novos recursos no banco de dados ou o disparo de comandos de processamento. No sistema de irrigação automatizado, o método POST é utilizado para:

1. **Autenticação de Usuários:** O formulário de login transmite as credenciais do operador via POST /login, recebendo como resposta o token JWT de autorização.
2. **Cadastro de Novos Agendamentos:** O diálogo modal envia um documento JSON via POST /agendamentos contendo a data, horários de início e término, solenoide selecionada e o operador responsável (como o agrônomo Thiago).
3. **Acionamento Manual de Atuadores:** O painel de controle permite ao operador forçar a alteração imediata do estado de uma válvula solenoide via POST/*solenoide?id=1&ativo=true*.

3.3. Métodos PUT, PATCH e DELETE: Atualização e Cancelamento

O método PUT é utilizado para substituição completa de um recurso existente, enquanto o PATCH é empregado para atualizações parciais de atributos específicos (como alterar apenas o horário de término de um agendamento). O método DELETE é utilizado para remover recursos do banco de dados, permitindo ao operador cancelar uma rotina de irrigação agendada antes do seu disparo automático.

4. ESTRUTURA DE MENSAGENS, CABEÇALHOS E PAYLOADS JSON

Uma mensagem HTTP é composta por três partes fundamentais: a linha de requisição

ou status, os cabeçalhos de metadados (*Headers*) e o corpo da mensagem (*Payload/Body*).

4.1. Cabeçalhos HTTP de Contexto e Segurança

Os cabeçalhos HTTP transportam informações sobre a natureza do cliente, o tipo de conteúdo aceito e as credenciais de segurança. Os principais cabeçalhos utilizados na comunicação da plataforma de irrigação incluem:

- *Content-Type: application/json:* Informa ao servidor que o corpo da requisição está formatado na linguagem JSON.
- *Accept: application/json:* Indica ao backend que a aplicação cliente espera receber dados estruturados em JSON como resposta.
- *Authorization: Bearer <token_jwt>:* Transporta o token de segurança em todas as requisições autenticadas emitidas pelo frontend web.

4.2. Formato JSON (JavaScript Object Notation)

O JSON consagrou-se como o formato padrão universal para troca de dados em sistemas IoT devido à sua leveza, simplicidade de parsing e legibilidade tanto por humanos quanto por máquinas. Em comparação a formatos verbosos como o XML, o JSON consome significativamente menos bytes de memória RAM e largura de banda de rede, um fator crítico para microcontroladores como o ESP32 operando em conexões de dados rurais.

No sistema de irrigação proposto, a resposta de sincronização do endpoint/*estado-geral* é codificada na seguinte estrutura JSON:

```
{
  "bomba":
  true,
  "solenoides":
  [
    { "id": 1, "ativo": true },
    { "id": 2, "ativo": false },
    { "id": 3, "ativo": false },
    { "id": 4, "ativo": false }
  ],
  "horario_servidor": "2026−09−29T07:30:00Z"
```

Essa estrutura permite que o nó de borda ESP32 identifique com precisão de milissegundos se a motobomba deve estar energizada e qual dos quatro solenoides (Setor Melão, Tomate, Pimentão ou Estufa) deve permanecer aberta.

5. CÓDIGOS DE STATUS HTTP E TRATAMENTO DE ERROS

O servidor backend responde a cada requisição enviando um código de status de três dígitos que informa o resultado do processamento. O entendimento desses códigos é

essencial para que o frontend web e os nós de borda consigam tratar exceções e reagir adequadamente a falhas.

Os códigos de status mais relevantes no sistema de irrigação incluem:

- **200 OK:** Indica que a requisição foi recebida, processada e atendida com sucesso pelo servidor Flask.
- **201 Created:** Confirma que uma nova rotina de irrigação foi cadastrada com sucesso no banco de dados MySQL.
- **400 Bad Request:** Informa que a requisição enviada possui parâmetros inválidos ou viola regras de salvaguarda (como tentar agendar uma rega com duração superior a 30 minutos).
- **401 Unauthorized:** Sinaliza que o operador não está autenticado ou que o token JWT expirou.
- **404 Not Found:** Indica que o endpoint ou recurso solicitado não existe no servidor.
- **500 Internal Server Error:** Alerta sobre uma falha interna no backend, como perda de conexão com o banco MySQL ou erro imprevisto na execução do script Python.

6. SÍNTESE DA COMUNICAÇÃO CLIENTE-SERVIDOR NA LAVOURA DIGITAL

A consolidação do modelo cliente-servidor através do protocolo HTTP REST estabelece o alicerce para a agricultura digital inteligente. Ao posicionar a inteligência de processamento e o banco de dados na camada superior e os nós de borda com a interface humana na camada inferior, o sistema atinge elevada robustez, desacoplamento e facilidade de manutenção.

Com o domínio conceitual das interfaces web (Capítulo 1), dos equipamentos e sensores de campo (Capítulo 2), dos circuitos eletrônicos (Capítulo 3) e do protocolo de comunicação HTTP (Capítulo 4), a infraestrutura tecnológica do sistema de irrigação automatizado encontra-se completamente fundamentada para aplicação na moderna agricultura de precisão.

REFERÊNCIAS

BERNERS-LEE, Tim; FIELDING, Roy; FRYSTYK, Henrik. **Hypertext Transfer Protocol – HTTP/1.0**. [S.l.], 1996. DOI: 10.17487/RFC1945.

FIELDING, Roy T.; NOTTINGHAM, Mark; RESCHKE, Julian. **HTTP Semantics**. [S.l.], 2022. DOI: 10.17487/RFC9110. Disponível em:<https://www.rfc-editor.org/rfc/rfc9110.html>.

PIERCE, Francis J.; NOWAK, Peter. **Aspects of Precision Agriculture**. [S.l.]: Advances in Agronomy, 1999. v. 67, p. 1–85. DOI: 10.1016/S0065-2113(08)60513-1.

STAFFORD, John V. **Implementing Precision Agriculture in the 21st Century**. London: Academic Press, 2000.

CAPÍTULO 5

FIRMWARE C++ PARA ESP32: ARQUITETURA DE CLIENTE MAGRO, SINCRONIZAÇÃO HTTP E ACIONAMENTO DE ATUADORES NA IRRIGAÇÃO IoT

1. INTRODUÇÃO E INSERÇÃO DO SISTEMA EMBARCADO NO AMBIENTE AGRÍCOLA

No paradigma da agricultura digital e da Internet das Coisas (Internet of Things – IoT), os microcontroladores de borda atuam como a ponte física indispensável entre o mundo digital (onde residem os algoritmos e servidores) e a lavoura (onde a água deve ser fisicamente aplicada ao solo). Enquanto os capítulos anteriores exploraram a construção das interfaces gráficas web (Capítulo 1), as especificações hidráulicas e eletromecânicas dos atuadores (Capítulo 2), a instrumentação eletrônica de controle e leitura (Capítulo 3) e o protocolo de comunicação HTTP REST (Capítulo 4), este capítulo dedica-se integralmente à inteligência embarcada que executa diretamente no microcontrolador ESP32.

O ESP32 é um sistema em chip (*System on a Chip* – SoC) de baixo custo, elevado desempenho e baixo consumo energético, dotado de conectividade Wi-Fi e Bluetooth nativas. No projeto hipotético de irrigação automatizada analisado nesta obra, o ESP32 fica instalado no painel elétrico de campo, posicionado estrategicamente próximo ao conjunto motobomba e ao coletor hidráulico de distribuição. Sua função principal é executar um firmware escrito na linguagem C++ que se conecta à rede sem fio da propriedade rural, consulta periodicamente uma API web remota e aciona os relés acoplados às válvulas solenoides e à motobomba principal.

Para que agrônomos, técnicos agrícolas e engenheiros de software trabalhem em harmonia, é preciso compreender que o código-fonte C++ que roda no microcontrolador não é apenas um script de controle: ele é o agente executor das decisões agronômicas. Cada linha de código relacionada ao consumo de dados via HTTP traduz a intenção de irrigar ou suspender a rega em respostas elétricas de 3,3 V nos pinos de saída (*General Purpose Input/Output* – GPIO).

2. A CONCEITUAÇÃO DE CLIENTE MAGRO (*THIN CLIENT*) SEGUNDO TANENBAUM

A estruturação do software embarcado no ESP32 segue os preceitos clássicos das arquiteturas distribuídas descritas pelo renomado cientista da computação Andrew S. Tanenbaum. Em sua classificação de modelos cliente-servidor, Tanenbaum estabelece a distinção entre clientes pesados (*Thick/Fat Clients*) e clientes magros (*Thin Clients*) com base na distribuição das camadas de apresentação, lógica de negócio e armazenamento de dados.

Um cliente pesado concentra localmente a inteligência de processamento, as regras de negócio e, frequentemente, a tomada de decisão. Caso o ESP32 fosse projetado como um cliente pesado, ele precisaria armazenar internamente as tabelas de coeficientes de

cultura (Kc), as equações de evapotranspiração de Penman-Monteith, os relógios de tempo real (RTC) com os horários de início e fim de irrigação para cada talhão, e os históricos de amostragem. No entanto, essa abordagem traria sérias limitações para cenários agrícolas rurais:

- **Complexidade de Manutenção e Atualização:** Qualquer alteração no manejo agronômico (como mudar o tempo de rega do setor do melão de 15 para 20 minutos) exigiria que um técnico fosse até o campo e reprogramasse fisicamente o microcontrolador via cabo USB ou atualizações de firmware over-theair (OTA).
- **Restrição de Recursos de Hardware:** Microcontroladores possuem memória RAM limitada (o ESP32 dispõe de 520 KB de SRAM interna). Executar cálculos complexos e armazenar grandes volumes de dados comprometeria a estabilidade do dispositivo.
- **Risco de Dessincronização:** Múltiplos microcontroladores em setores distintos da fazenda poderiam operar com relógios descalibrados ou regras desatualizadas, gerando conflitos no uso da bomba hidráulica principal.

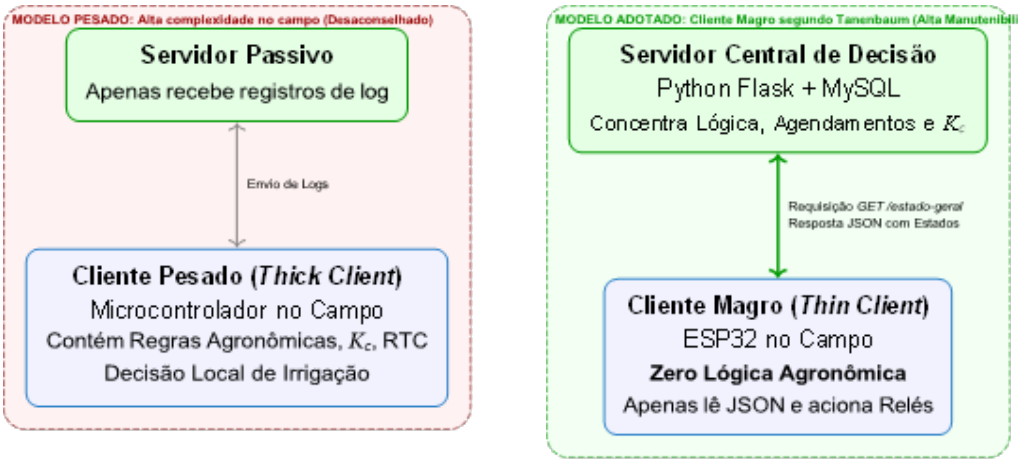
Para contornar as limitações de hardware, o firmware desenvolvido espelha o modelo de **Cliente Magro** (*Thin Client*) (TANENBAUM; STEEN, 2017) adaptado à arquitetura de borda e nuvem (SHI; CAO et al., 2016; SATYANARAYANAN, 2017). Em uma versão simplificada, o dispositivo de borda realiza apenas a interface de entrada/-saída, enquanto toda a tomada de decisão agronômica reside no servidor central. No entanto, é fundamental destacar que sistemas IoT profissionais raramente utilizam dispositivos de borda desprovidos de lógica autônoma. Visando resiliência, a computação de borda moderna orienta a manutenção de regras mínimas de segurança local, cronômetros de vigilância (*watchdogs*), limites operacionais e lógicas elementares no microcontrolador, garantindo que o sistema atue proativamente mesmo diante de prolongadas falhas na rede de acesso global (SHI; DUSTDAR, 2016; BAZZI et al., 2023).

Na prática da irrigação IoT, o ESP32 operando como cliente magro não sabe se o solo está seco ou úmido, não calcula a lâmina de água e não possui conhecimento sobre agendamentos futuros. O ESP32 simplesmente limita-se a:

1. Conectar-se à rede Wi-Fi local;
2. Disparar uma requisição HTTP do tipo GET para o servidor central Flask a cada 5 segundos;
3. Receber a resposta contendo a decisão já tomada pelo servidor em formato JSON;
4. Alterar o estado dos pinos GPIO (LIGADO/DESLIGADO) de acordo com os valores booleanos recebidos.

A Figura 1 ilustra a comparação conceitual entre as abordagens de cliente pesado e cliente magro de Tanenbaum aplicadas à automação agrícola.

Figura 15: Comparação arquitetural segundo a taxonomia de Tanenbaum: no modelo de Cliente Magro adotado, o ESP32 limita-se a executar acionamentos elétricos instruídos pelo servidor central.



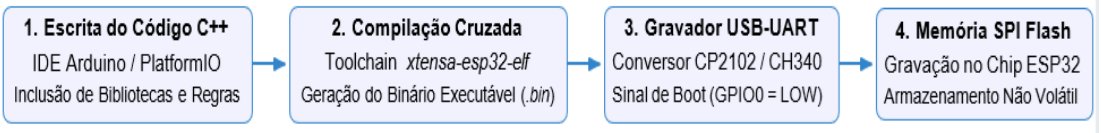
Essa descentralização traz vantagens expressivas para o produtor rural: se o agrônomo Thiago decidir reconfigurar todos os parâmetros de irrigação do cultivo de tomate, a alteração é efetuada instantaneamente na interface web. No próximo ciclo de consulta (5 segundos depois), o ESP32 receberá as novas ordens de acionamento sem necessidade de qualquer intervenção técnica no hardware de campo.

3. FLUXO DE DESENVOLVIMENTO, COMPILAÇÃO E GRAVAÇÃO (FLASHING) NO ESP32

Para que o código C++ escrito no computador passe a executar de maneira autônoma no chip ESP32 instalado na fazenda, ele deve submeter-se a um fluxo estruturado de engenharia de software embarcado. Esse ciclo compreende as etapas de escrita, compilação cruzada, transferência serial para a memória Flash e inicialização do microcontrolador.

A Figura 2 apresenta a esteira completa do fluxo de subida (flashing) do programa no ESP32.

Figura 16: Fluxo de compilação cruzada e gravação do firmware C++ no microcontrolador ESP32.



As etapas do processo de gravação desdobram-se conforme detalhado a seguir:

1. **Ambiente de Desenvolvimento (IDE):** O programador de TI escreve o código C++ utilizando um ambiente como a Arduino IDE ou o VS Code com a extensão PlatformIO. O código faz referência às bibliotecas nativas de comunicação sem fio e serialização de dados.
2. **Compilação Cruzada (*Cross-Compilation*):** Como o computador de desenvolvimento

possui arquitetura x86/x64 e o ESP32 utiliza microarquitetura Xtensa LX6 de 32 bits, a IDE utiliza um compilador cruzado (*xtensa-esp32-elf-g++*). O compilador traduz as instruções em C++ para linguagem de máquina executável e gera um arquivo binário comprimido com extensão .bin.

3. **Comunicação Serial e Protocolo de Bootloader:** O computador é conectado à placa ESP32 por meio de um cabo USB. Um chip conversor USB-para-UART (como o CP2102 ou CH340 instalado na placa) converte os pacotes USB em sinais seriais TTL (transmissão TX e recepção RX). Para iniciar a gravação, o circuito de autorreset coloca o pino GPIO0 em nível lógico baixo (0 V) enquanto aplica um pulso no pino EN (Reset), forçando a ROM do ESP32 a entrar no modo de gravação (Download Bootloader).
4. **Gravação na Memória SPI Flash:** O programa utilitário (esptool.py) envia os blocos do arquivo binário através da interface serial a uma taxa de transmissão típica de 921.600 baud. Os dados são gravados permanentemente na memória Flash externa do ESP32. Por ser uma memória não volátil, o código-fonte permanece salvo mesmo quando a alimentação elétrica da fazenda é desligada.
5. **Inicialização Autônoma:** Concluída a gravação, o pino EN é liberado, o ESP32 reinicia em modo de execução normal, carrega o código da Flash para a memória SRAM e executa automaticamente a função setup(), seguida do loop infinito loop().

4. DEPENDÊNCIAS DE SOFTWARE E BIBLIOTECAS DA PLATAFORMA EMBARCADA

O funcionamento do cliente magro ESP32 fundamenta-se em três bibliotecas principais do ecossistema Arduino/ESP32. O entendimento do papel de cada uma dessas dependências é essencial para a correta compilação e operação do sistema:

4.1. Biblioteca *WiFi.h*

A biblioteca *WiFi.h* é a pilha de protocolos de comunicação sem fio nativa do núcleo ESP32 para Arduino. Ela abstrai a complexidade do hardware de radiofrequência de 2,4 GHz e fornece métodos em C++ para gerenciar o estado da conexão de rede. Suas principais responsabilidades no sistema de irrigação incluem:

- Configurar o modo de operação do chip como Estação Wi-Fi (*WIFI_STA*), permitindo que o ESP32 se conecte ao roteador da fazenda como um dispositivo cliente.
- Negociar o aperto de mão (*handshake*) de segurança utilizando criptografia WPA2-PSK.
- Gerenciar a atribuição dinâmica de endereço IP via DHCP (*Dynamic Host Configuration Protocol*).
- Fornecer a função de sondagem de estado *WiFi.status()*, que retorna a constante

WL_CONNECTED quando a conexão com o ponto de acesso está ativa e funcional.

4.2. Biblioteca *HTTPClient.h*

A biblioteca *HTTPClient.h* implementa a camada de aplicação HTTP/HTTPS sobre a pilha TCP/IP do ESP32. Ela permite que o microcontrolador funcione como um cliente web capaz de abrir soquetes de rede, formatar requisições HTTP REST e ler respostas transmitidas por servidores remotos. Dentre suas funções fundamentais, destacam-se:

- Estabelecer a conexão com a URL de destino através do método *begin(serverName)*.
- Executar transações HTTP como requisições GET (*http.GET()*) ou POST (*http.POST()*).
- Negociar a camada de segurança TLS/SSL para conexões criptografadas HTTPS (porta 443).
- Capturar os códigos de status de retorno HTTP (como 200 OK ou 404 Not Found) e extrair o corpo da resposta em texto simples através do método *getString()*.

4.3. Biblioteca *ArduinoJson.h*

A biblioteca *ArduinoJson* (desenvolvida por Benoît Blanchon) é a biblioteca padrão para serialização e desserialização de documentos JSON em sistemas embarcados. Devido à rigidez de memória dos microcontroladores, a *ArduinoJson* se destaca por gerenciar a memória de forma extremamente eficiente, evitando fragmentação na SRAM. Suas atribuições no firmware incluem:

- Alocar um documento de memória otimizado através da classe *JsonDocument*.
- Converter a string texto obtida do servidor em uma estrutura de dados de árvore pesquisável em C++ por meio do método *deserializeJson(doc, payload*).
- Validar a integridade estrutural da mensagem recebida através do objeto de verificação de erros *DeserializationError.*
- Permitir o acesso a arrays (*JsonArray*) e objetos (*JsonObject*) com rápida conversão de tipos de dados (booleanos, inteiros e strings).

5. APRESENTAÇÃO E ANÁLISE DO CÓDIGO-FONTE C++

O código C++ completo responsável por transformar o ESP32 no cliente magro de controle de irrigação é apresentado no arquivo de listagem a seguir.

Listing 1: Código C++ completo para o microcontrolador ESP32: sincronização via HTTP e acionamento de relés.

```
#include <WiFi.h> #include
<HTTPClient.h> #include
<ArduinoJson.h>

#define LIGADA HIGH #define
DESLIGADA LOW #define
BOMBA 22

#define SOLENOIDE_1 23
#define SOLENOIDE_2 21
#define SOLENOIDE_3 32
#define SOLENOIDE_4 18


const char *ssid = "Tenda_9519E8";
const char *password = "able7436";
const char *serverName = "https://sergiosouzanovak.com.br/backend-irrigacao/estado-geral";


const int reles[5] = {SOLENOIDE_1, SOLENOIDE_2, SOLENOIDE_3, SOLENOIDE_4, BOMBA};


void desligarTudo() {
  for (int i = 0; i < 5; i++) {
    digitalWrite(reles[i], DESLIGADA);
  }
}


void setup() {
  Serial.begin(115200);


  for (int i = 0; i < 5; i++) { pinMode(reles[i],
    OUTPUT); digitalWrite(reles[i],
```

```
36    while (WiFi.status() != WL_CONNECTED) {
37      delay(500);
38      Serial.print(".");
39    }
40
41    Serial.println("\nConectado!");
42  }
43
44  void loop() {
45
46    if (WiFi.status() == WL_CONNECTED) {
47
48      HTTPClient http;
49      http.begin(serverName);
50
51      int httpResponseCode = http.GET();
52
53      if (httpResponseCode > 0) {
54
55        String payload = http.getString();
56        Serial.println("JSON: " + payload);
57
58        JsonDocument doc;
59        DeserializationError error = deserializeJson(doc, payload);
60
61        if (error) {
62
63          Serial.println("erro requisicao");
64          desligarTudo();
65
66        } else {
67
68          JsonArray solenoides = doc["solenoides"].as<JsonArray>();
69
70          for (int i = 0; i < 4; i++) {
71
72            int idSolenoid = i + 1;
73            bool ativo = false;
74
75            for (JsonObject sol : solenoides) {
76              if (sol["id"].as<int>() == idSolenoid) {
77                ativo = sol["ativo"].as<bool>();
```

```
78                    break;
79                  }
80                }
81
82                digitalWrite(reles[i], ativo ? LIGADA : DESLIGADA);
83
84                Serial.printf(
85                  "Solenoide %d (GPIO %d): STATUS %s\n",
86                  idSolenoid,
87                  reles[i],
88                  ativo ? "LIGADA" : "DESLIGADA"
89                );
90              }
91
92              bool bombaAtiva = doc["bomba"].as<bool>();
93              digitalWrite(reles[4], bombaAtiva ? LIGADA : DESLIGADA);
94
95              Serial.printf("Bomba (GPIO %d): STATUS %s\n", BOMBA, bombaAtiva ? "LIGADA" : "DESLIGADA");
96            }
97
98          } else {
99
100           Serial.println("erro requisicao");
101           desligarTudo();
102         }
103
104         http.end();
105
106       } else {
107
108         Serial.println("erro requisicao");
109         desligarTudo();
110       }
111
112     delay(5000);
113   }
```

6. ESTRUTURA DE CONSTANTES, MAPEAMENTO DE GPIOS E PARAMETRIZAÇÃO DE REDE

A primeira seção do código (linhas 1 a 16 da Listagem 1) estabelece as fundações operacionais do firmware, mapeando os pinos elétricos do microcontrolador e registrando os parâmetros de conectividade.

6.1. Diretivas de Mapeamento de Pinos *(#define)*

O código utiliza diretivas do pré-processador C++ para associar nomes amigáveis às constantes de estado e aos pinos físicos de entrada e saída (GPIOs):

- *#define LIGADA HIGH* e *#define DESLIGADA LOW:* Definem que o estado LIGADA corresponde ao nível lógico alto (3,3 V na saída do ESP32) e o estado DES-LIGADA ao nível lógico baixo (0 V). Essa abstração evita erros de lógica e facilita a leitura do código por profissionais de agronomia.
- *#define BOMBA 22*: Associa o controle da motobomba principal ao pino digital GPIO 22.
- *#define SOLENOIDE_1 23, SOLENOIDE_2 21, SOLENOIDE_3 32 e SOLENOIDE_4 18:* Mapeiam os quatro setores hidráulicos de irrigação da propriedade às respetivas portas GPIO do ESP32.

6.2. Parâmetros da Rede Wi-Fi e Endpoint HTTPS

As constantes de ponteiro de caractere armazenam as credenciais de autenticação e a URL do servidor central:

- *const char *ssid* = "*Tenda_9519E8*": O identificador do ponto de acesso Wi-Fi instalado no galpão agrícola da fazenda.
- *const char *password* = "*able7436*": A chave de segurança para associação à rede sem fio.
- *const char *serverName* = https://sergiosouzanovak.com.br/backend-irrigacao/estado-geral: O endereço HTTPS completo da API REST que o ESP32 consultará a cada ciclo para obter o estado dos atuadores.

6.3. Vetor de Reorganização de Atuadores

A instrução *const int reles[5] = {SOLENOIDE_1, SOLENOIDE_2, SOLENOIDE_3, SO-LENOIDE_4, BOMBA}* agrupa as constantes de pinos em uma estrutura indexada de vetor (array). Essa abordagem é uma excelente prática de programação embarcada, pois permite que o firmware percorra todos os atuadores da lavoura utilizando laços de repetição (for), reduzindo drasticamente o tamanho do código compilado e simplificando a manutenção.

7. MECANISMO DE SEGURANÇA E SALVAGUARDA HIDRÁULICA: A FUNÇÃO *desligarTudo()*

Dentre todos os blocos do firmware, a função *desligarTudo()* (linhas 18 a 22 da Listagem 1) reveste-se de importância crítica para a segurança agronômica e patrimonial da propriedade rural.

```
void desligarTudo() {
  for (int i = 0; i < 5; i++) {
    digitalWrite(reles[i], DESLIGADA);
  }
}
```

Seu funcionamento consiste em iterar pelos cinco elementos do vetor de relés e aplicar o nível lógico *DESLIGADA* (0 V) a cada uma das portas GPIO, forçando a abertura imediata dos contatos dos relés e desenergizando os solenoides e a motobomba.

7.1. O Conceito de Falha Segura (*Fail-Safe*) na Agricultura

Na engenharia de controle e automação, um sistema é classificado como Falha Segura (*Fail-Safe*) quando, diante de qualquer anomalia ou colapso na infraestrutura, ele transiciona automaticamente para um estado passivo inofensivo, impedindo catástrofes físicas ou econômicas.

Na irrigação agrícola automatizada, a ocorrência de falhas é um evento inevitável devido às intempéries do ambiente rural:

- O sinal da rede Wi-Fi pode sofrer degradação temporária por interferência atmosférica ou obstrução por máquinas agrícolas;
- O servidor backend Flask pode ficar inacessível por queda no fornecimento de energia da cidade;
- O roteador da fazenda pode reiniciar por sobreaquecimento;
- O pacote de dados JSON retornado pela API pode ser corrompido durante a transmissão.

Se o microcontrolador ESP32 não possuísse o mecanismo de falha segura da função *desligarTudo()*, qualquer queda de conexão corrida no momento em que a bomba estivesse ligada faria com que o sistema mantivesse os relés energizados indefinidamente. As consequências agronômicas e financeiras dessa omissão seriam severas:

1. **Encharcamento e Anoxia Radicular:** A aplicação ininterrupta de água saturaria o perfil do solo, expulsando o oxigênio dos macroporos, provocando a asfixia

do sistema radicular das plantas e o surgimento de doenças fúngicas (como *Phytophthora* spp.).

2. **Rompimento de Tubulações Hidráulicas:** Se os solenoides fossem fechados mas a motobomba permanecesse ligada sem vazão, a pressão manométrica da rede ultrapassaria os limites dos tubos de PVC, provocando o estouro de adutoras.
3. **Cavitação e Queima da Motobomba:** O funcionamento continuado da motobomba sem circulação renovada de água causaria o superaquecimento do fluido interno, destruição dos selos mecânicos e queima do enrolamento elétrico do motor de 5,5 CV.
4. **Desperdício Hídrico e Energético:** Dezenas de milhares de litros de água e quilowatts-hora de energia elétrica seriam consumidos inultilmente, elevando os custos operacionais do produtor.

Na lógica implementada para fins didáticos, a função *desligarTudo()* é acionada como uma aproximação de falha segura em todos os caminhos de exceção do firmware: perda de Wi-Fi, falha HTTP ou JSON corrompido (INTERNATIONAL ELECTROTE-CHNICAL COMMISSION, 2016). Contudo, em cenários industriais e agrícolas de grande porte, o conceito de falha segura deve obrigatoriamente derivar de uma análise formal de perigos e matriz de riscos (LEVESON, 2011). Desligar todos os atuadores subitamente nem sempre representa o estado mais seguro; certos processos exigem válvulas normalmente abertas para alívio de pressão hidráulica ou intertravamentos específicos previstos em normas de segurança funcional (INTERNATIONAL ELECTROTECHNI-CAL COMMISSION, 2010; INTERNATIONAL ORGANIZATION FOR STANDAR-DIZATION, 2010). Assim, embora a abordagem de contingência total proteja o sistema simplificado aqui concebido, sua escalabilidade exige a avaliação criteriosa de modos degradados e tolerância a falhas (BENNETT, 1994).

8. PROCEDIMENTO DE INICIALIZAÇÃO: A FUNÇÃO *setup()*

A função *setup()* (linhas 24 a 40 da Listagem 1) é executada exatamente uma única vez, logo que o ESP32 é energizado ou após a ocorrência de um sinal de reset. Ela é responsável por preparar a infraestrutura interna do microcontrolador antes da entrada no ciclo de operação continuada.

1. **Inicialização do Monitor Serial:** A instrução *Serial.begin(115200*) configura a porta de comunicação serial assíncrona para operar a uma taxa de transmissão de 115.200 bits por segundo (baud rate). Isso possibilita que o técnico de TI conecte um computador ao ESP32 e monitore em tempo real os relatórios de diagnóstico emitidos pelo firmware.
2. **Configuração de Direção dos Pinos (*pinMode*):** O laço for percorre as portas digi-

tais associadas aos relés e executa a função *pinMode(reles[i], OUTPUT*), instruindo o hardware interno do ESP32 a configurar esses pinos como saídas digitais capazes de fornecer corrente para os optoacopladores das placas de relé.

3. **Garantia de Estado Inicial Desenergizado:** Imediatamente após definir os pinos como saída, a instrução *digitalWrite(reles[i], DESLIGADA)* força todos os relés ao nível 0 V. Isso evita o acionamento indesejado dos motores durante a etapa de energização do painel elétrico.
4. **Associação Wi-Fi e Espera Bloqueante:** A chamada *WiFi.begin(ssid, password)* dispara a solicitação de conexão à rede sem fio da propriedade. Em seguida, o programa entra em um laço de repetição bloqueante while(WiFi.status()!=WL_ CONNECTED), no qual o ESP32 aguarda a confirmação da conexão, emitindo um ponto na interface serial a cada 500 milissegundos (*delay(500)*). Assim que a associação é concluída, a mensagem "*Conectado*!" é exibida na tela.

9. O CICLO INFINITO DE POLLING E CONSUMO DA API: A FUNÇÃO *loop()*

Após o término da função *setup()*, o controle de execução transita para a função *loop()* (linhas 42 a 110 da Listagem 1), que é executada continuamente de forma repetitiva enquanto o microcontrolador permanecer energizado.

A Figura 3 ilustra a máquina de estados completa e o fluxograma de decisões executado dentro da função *loop()*.

9.1. Verificação de Conectividade Wi-Fi

No início de cada iteração, a *instrução if (WiFi.status() == WL_CONNECTED)* testa se a conexão sem fio permanece ativa. Se o roteador tiver caído ou a placa tiver perdido o sinal, o programa desvia diretamente para o bloco else, emitindo a mensagem de erro no monitor serial e executando a função de segurança *desligarTudo().*

9.2. Disparo da Requisição HTTP GET

Estando a rede Wi-Fi operacional, o firmware instancia um objeto da classe *HTTPClient* e associa o endereço HTTPS de destino através de *http.begin(serverName).* A chamada bloqueante *int httpResponseCode = http.GET()* faz com que o ESP32 monte o pacote HTTP, abra a conexão TCP na porta 443, negocie a criptografia TLS e aguarde o retorno do servidor central Flask.

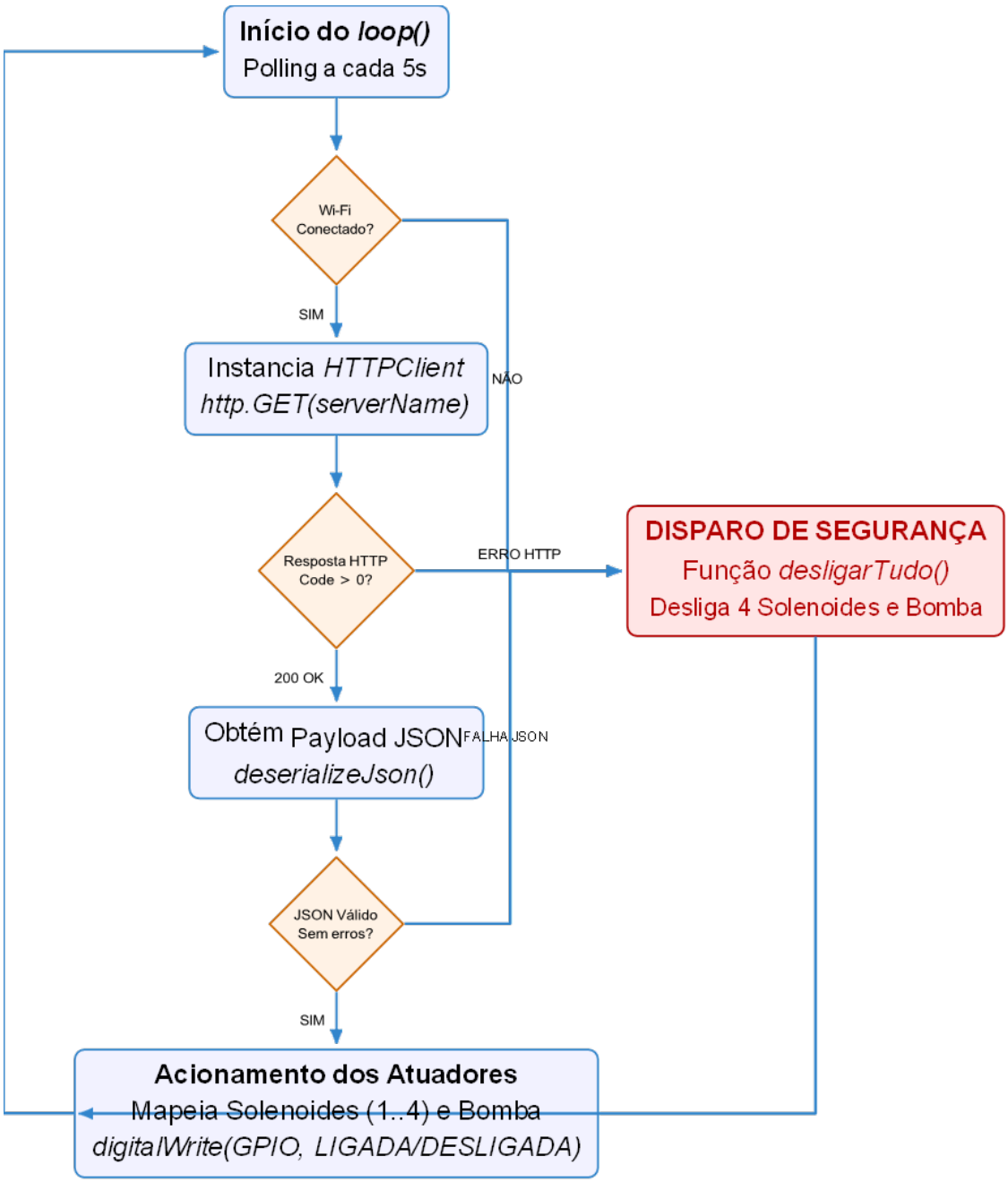
Se a comunicação for bem-sucedida, o código de status retornado é maior que zero (tipicamente 200 OK). Caso contrário (se o servidor estiver fora do ar ou o nome de domínio não puder ser resolvido pelo DNS), o código retornado é um valor negativo e o firmware aciona imediatamente o bloco de segurança *desligarTudo().*

9.3. Desserialização e Parsing do Documento JSON

Ao confirmar o recebimento de uma resposta HTTP válida, o ESP32 extrai o texto do

corpo da mensagem por meio de String payload = http.getString(). Em seguida, o firmware aloca uma instância de JsonDocument e executa o método deserializeJson(doc,

Figura 17: Fluxograma de decisões e salvaguardas da função *loop()* do ESP32.



O parâmetro de verificação de erro *DeserializationError* error analisa se a string recebida obedece rigorosamente à sintaxe JSON. Se o pacote tiver chegado incompleto ou corrompido, a condição if (error) é satisfeita, acionando a função *desligarTudo()* para evitar que dados truncados levem a acionamentos erráticos de solenoides no campo.

9.4. Mapeamento Dinâmico dos Setores e Acionamento Elétrico

Se a desserialização JSON for concluída com sucesso, o firmware

inicia o processo de interpretação dos dados recebidos. A resposta típica emitida pela API do sistema de irrigação apresenta o seguinte formato:

```
{
  "bomba": true,
  "solenoides": [
    { "id": 1, "ativo": true },
    { "id": 2, "ativo": false },
    { "id": 3, "ativo": false },
    { "id": 4, "ativo": false }
  ]
}
```

O código acessa o vetor de solenoides através da instrução *JsonArray solenoides = doc[“solenoides”*].as<JsonArray>(). Em seguida, um laço de repetição for (int i = 0; i < 4; i++) percorre as quatro válvulas cadastradas:

```
for (int i = 0; i < 4; i++) {
  int idSolenoid = i + 1;
  bool ativo = false;

  for (JsonObject sol : solenoides) {
    if (sol["id"].as<int>() == idSolenoid) {
      ativo = sol["ativo"].as<bool>();
      break;
    }
  }

  digitalWrite(reles[i], ativo ? LIGADA : DESLIGADA);
}
```

Para cada setor da lavoura (1 a 4), o algoritmo busca a correspondência exata do atributo “*id*” dentro do array JSON. Ao localizar o objeto correspondente, a variável booleana *ativo* recebe o valor presente na chave “*ativo*”.

A instrução ternária *digitalWrite(reles[i], ativo ? LIGADA : DESLIGADA*) aplica diretamente a decisão: se ativo for verdadeiro (*true*), a saída GPIO física do ESP32 é colocada em 3,3 V (*HIGH*), acionando a bobina do relé e abrindo a válvula solenoide correspondente na tubulação de irrigação. Caso contrário, a saída é mantida em 0 V (*LOW*), fechando a válvula.

9.5. Acionamento da Motobomba Principal e Intervalo de Polling

Após atualizar o estado das quatro solenoides, o firmware lê o atributo booleano correspondente à bomba principal:

```
1 bool bombaAtiva = doc["bomba"].as<bool>();
2 digitalWrite(reles[4], bombaAtiva ? LIGADA : DESLIGADA);
```

A instrução aplica a leitura diretamente na porta GPIO 22 *(reles[4]),* energizando ou desenergizando a bobina da contatora elétrica de potência que alimenta a motobomba de 5,5 CV.

Por fim, a função *http.end()* encerra formalmente a conexão do soquete HTTP, liberando os recursos de memória RAM do ESP32, e a instrução *delay(5000)* suspende a execução por 5.000 milissegundos (5 segundos) antes que o próximo ciclo da função *loop()* seja iniciado.

10. CONSIDERAÇÕES PRÁTICAS E ENGENHARIA DE IMPLANTAÇÃO NO CAMPO

A implementação com sucesso de um cliente magro ESP32 na agricultura de precisão exige a observância de boas práticas de engenharia eletrônica e de redes rurais:

- **Estabilidade de Alimentação Elétrica:** Motores e solenoides geram ruídos eletromagnéticos e surtos de tensão ao serem acionados. O ESP32 deve ser alimentado por uma fonte chaveada regulada independente de 5 V / 2 A, acompanhada de capacitores de desacoplamento e diodos de roda livre (*flyback*) instalados nas bobinas dos relés para evitar autorresets provocados por transientes elétricos.
- **Qualidade de Sinal Sem Fio (RSSI):** Como o cliente magro depende totalmente da comunicação com o servidor para irrigar, o painel elétrico de campo deve possuir antena Wi-Fi externa com ganho adequado (ex: antena omnidirecional de 5 dBi) para garantir uma relação sinal-ruído robusta em relação ao roteador do galpão.
- **Certificados de Segurança em Conexões HTTPS:** A utilização da URL com pro-tocolo criptografado *https://* exige atenção ao gerenciamento de certificados SSL/-TLS no ESP32. Em sistemas de produção, deve-se carregar a chave do certificado de autoridade certificadora (CA) ou utilizar validação de impressão digital (*finger-print*) para impedir ataques de interceptação na rede sem fio da fazenda.

11. SÍNTESE DO CAPÍTULO E INTEGRAÇÃO DO SISTEMA DE IRRIGAÇÃO

O desenvolvimento do firmware C++ para o ESP32 consolida o modelo de arquitetura distribuída de cliente magro proposto por Tanenbaum. Ao transferir a inteligência agronômica, o cálculo das lâminas de água e as tabelas de agendamento para o servidor central Flask/MySQL e manter o microcontrolador dedicado exclusivamente à sincronização HTTP e acionamento elétrico dos relés, o sistema atinge os mais elevados padrões de manutenibilidade, segurança e flexibilidade operacional.

Com este capítulo, encerra-se a fundamentação completa da obra: a integração harmônica entre as interfaces gráficas web (Capítulo 1), a infraestrutura de sensores e atuadores (Capítulo 2), os circuitos eletrônicos de instrumentação e acionamento (Capítulo 3), o protocolo de comunicação HTTP REST (Capítulo 4) e o código embarcado em C++ no microcontrolador ESP32 (Capítulo 5). Essa união entre a Ciência da Computação e a Engenharia Agronômica estabelece as bases definitivas para a construção de sistemas de irrigação automatizados de alta performance na lavoura digital.

REFERÊNCIAS

BAZZI, Alessandro et al. (Ed.). **Device-Edge-Cloud Continuum: Paradigms, Architectures and Applications**. [S.l.]: Springer, 2023. DOI: 10.1007/978-3-031-42194-5.

BENNETT, Stuart. **Real-Time Computer Control: An Introduction**. New York: Prentice Hall, 1994.

INTERNATIONAL ELECTROTECHNICAL COMMISSION. **Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems**. [S.l.: s.n.], 2010.

INTERNATIONAL ELECTROTECHNICAL COMMISSION. **Safety of Machinery: Electrical Equipment of Machines – Part 1: General Requirements**. [S.l.: s.n.], 2016.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **Safety of Machinery: General Principles for Design – Risk Assessment and Risk Reduction**. [S.l.: s.n.], 2010.

LEVESON, Nancy G. **Engineering a Safer World: Systems Thinking Applied to Safety**. Cambridge: MIT Press, 2011.

SATYANARAYANAN, Mahadev. The emergence of edge computing. **Computer**, v. 50, n. 1, p. 30–39, 2017. DOI: 10.1109/MC.2017.9.

SHI, Weisong; CAO, Jie et al. Edge computing: Vision and challenges. **IEEE Internet of Things Journal**, v. 3, n. 5, p. 637–646, 2016. DOI: 10.1109/JIOT.2016.2579198.

SHI, Weisong; DUSTDAR, Schahram. The promise of edge computing. **Computer**, v. 49, n. 5, p. 78–81, 2016. DOI: 10.1109/MC.2016.145.

TANENBAUM, Andrew S.; STEEN, Maarten van. **Distributed Systems: Principles and Paradigms**. 3. ed. Upper Saddle River: Pearson, 2017.

CAPÍTULO 6

O FRONTEND NA AUTOMATIZAÇÃO AGRÍCOLA: ARQUITETURA SPA, TECNOLOGIAS WEB E DECOUPLING DO BACKEND

1. INTRODUÇÃO AO FRONTEND PARA O LEITOR AGRÔNOMO: ANALO-GIA COM A ESTRUTURA VEGETAL

Para um engenheiro agrônomo, produtor rural ou técnico agrícola, a terminologia utilizada no desenvolvimento de sistemas computacionais pode parecer, à primeira vista, abstrata e distante da realidade de campo. Termos como *Frontend, HTML, CSS, JavaScript e Single Page Application* (SPA) frequentemente soam como um idioma exclusivo dos profissionais da Tecnologia da Informação (TI). No entanto, a compreensão dessas tecnologias torna-se surpreendentemente simples e intuitiva quando estabelecemos uma analogia direta com a anatomia e a fisiologia das plantas cultivadas na lavoura.

Em um sistema automatizado de irrigação baseada em Internet das Coisas (IoT), o termo **Frontend** refere-se a tudo aquilo que o usuário vê e interage diretamente na tela do seu computador, tablet ou smartphone. Ele é a interface visual — o painel de controle, os botões de ligar e desligar motobombas, os gráficos de umidade do solo e os formulários de agendamento de rega.

Para compreender como o Frontend é construído internamente na aplicação hipotética localizada na pasta /sistema/frontend, é útil analisar as três linguagens fundamentais da Web sob a ótica da biologia vegetal:

1. **HTML (*HyperText Markup Language*) – A Anatomia e o Esqueleto da Aplicação:** O HTML equivale à estrutura morfológica da planta (o esqueleto formado pelas raízes, caule, ramos e parênquima foliar). Ele não possui cores elaboradas ou movimentos ativos, mas define a existência e a posição de cada elemento estrutural. No sistema de irrigação, o HTML declara: "*aqui haverá um título principal, aqui haverá um botão para acionar a solenoide do setor do tomate, e aqui haverá uma tabela com os horários agendados*".
2. **CSS (*Cascading Style Sheets*) – A Estética, Cutícula e Pigmentação:** O CSS equivale à aparência externa, às cores e às texturas da planta (a pigmentação verde da clorofila nas folhas, o tom vibrante das flores, a espessura da cutícula cerosa e a disposição espacial dos ramos). Sem o CSS, o HTML seria exibido como um documento de texto plano, sem atrativo visual. O CSS é responsável por definir que o botão da motobomba terá cor azul vibrante, cantos arredondados, fontes tipográficas modernas (como a fonte Roboto) e um layout responsivo que se adapta perfeitamente tanto à tela de um monitor de 27 polegadas no escritório quanto à tela de um celular em pleno campo.
3. **JavaScript (JS) / TypeScript – A Fisiologia e os Reflexos Ativos:** O JavaScript equivale à fisiologia ativa da planta (o fluxo de seiva bruta e elaborada nos vasos

condutores de xilema e floema, a abertura e fechamento dos estômatos em resposta à radiação solar e os movimentos de tropismo). O JavaScript confere "vida" e interatividade ao esqueleto HTML e ao visual CSS. Quando o agrônomo clica no botão "Acionar Irrigação", é o JavaScript que captura esse evento de toque, valida se o usuário preencheu os campos corretamente, dispara uma mensagem invisível via rede sem fio para a API backend e atualiza o indicador visual da tela de "DESLI-GADO" para "LIGADO" sem piscar a página.

2. O CONCEITO E AS VANTAGENS DA ARQUITETURA SPA (*SINGLE PAGE APPLICATION*)

Na evolução da história da computação Web, os sistemas evoluíram de páginas tradicionais estáticas para a moderna arquitetura conhecida como **SPA** (*Single Page Application*, ou Aplicação de Página Única). Compreender a diferença entre essas duas abordagens é fundamental para entender o motivo pelo qual a aplicação de irrigação agrícola foi desenvolvida utilizando o framework Angular.

2.1. O Modelo Web Tradicional vs. O Modelo SPA

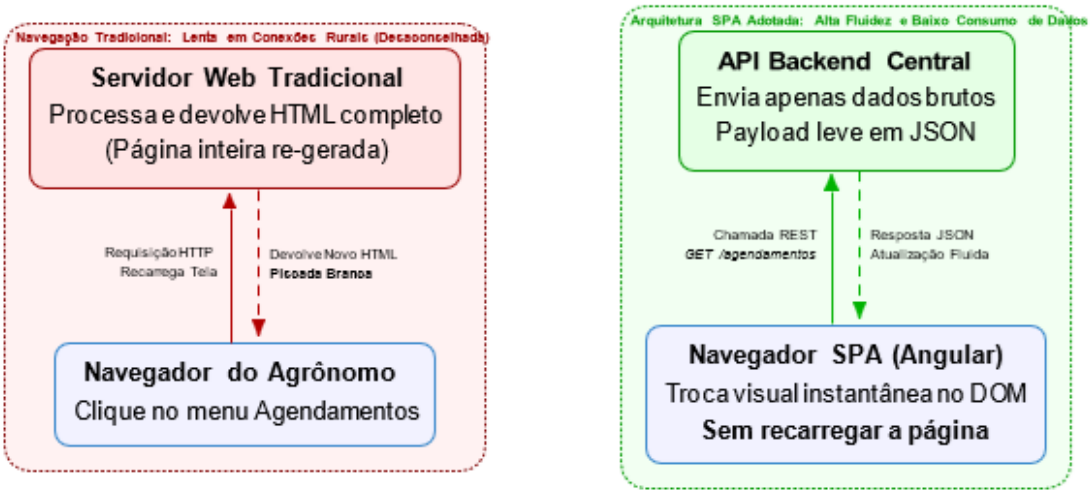
Em um website tradicional do início dos anos 2000, cada interação do usuário (como clicar no menu "Agendamentos" ou avançar para a página "Status da Lavoura") forçava o navegador a descartar completamente a tela atual, solicitar um novo documento HTML inteiro ao servidor remoto, aguardar o download de todos os arquivos de estilo e renderizar a página do zero. Essa abordagem resultava na famosa "piscada branca" na tela e em longos tempos de espera.

Em um ambiente agrícola rural — onde a conectividade de dados via redes móveis 3G/4G ou internet via satélite pode apresentar alta latência e largura de banda restrita—, o modelo tradicional torna a gestão operacional lenta, frustrante e propensa a falhas de carregamento.

Por outro lado, a arquitetura **SPA** revolucionou a experiência do usuário. Em uma Aplicação de Página Única, o navegador de internet carrega o arquivo estrutural principal (*index.html*) e os pacotes de código visual e lógico apenas uma única vez, no primeiro acesso do usuário. A partir desse momento, quando o agrônomo navega entre as seções do sistema (do painel de status para o formulário de novo agendamento, por exemplo), o navegador **não recarrega a página**. Em vez disso, o código JavaScript simplesmente remove os componentes visuais antigos da tela e insere os novos componentes instantaneamente, consumindo apenas pequenos pacotes de dados no plano de fundo.

A Figura 1 apresenta o diagrama comparativo entre o fluxo de navegação tradicional (com recarregamentos completos) e o fluxo SPA (com trocas dinâmicas de componentes e atualizações assíncronas em JSON).

Figura 18: Comparativo visual de navegação: o modelo SPA adotado elimina recarregamentos de página e consome apenas pequenos dados JSON via API.



2.2. Por que Utilizar SPA no Monitoramento Agrícola?

Para o produtor rural e para a equipe de agronomia, a adoção da arquitetura SPA traz benefícios práticos diretos:

- **Economia de Tráfego de Dados:** Como a estrutura visual já está salva no navegador, o sistema não gasta a franquia de dados da internet rural rebaixando os mesmos logotipos, menus e arquivos de estilo a cada clique. Apenas alguns poucos bytes de dados numéricos (como leituras de umidade e status de válvulas) trafegam pela rede.
- **Resposta Visual Instantânea:** Ao clicar para ligar um setor de irrigação, o operador recebe feedback imediato no painel, sem interromper sua navegação ou perder o contexto do mapa da fazenda.
- **Operação Offline e Tolerância a Oscilações:** Caso o sinal de internet da fazenda oscile por alguns segundos enquanto o agrônomo navega pelo painel, a aplicação SPA não exibe uma tela de erro de conexão completa. Ela mantém a interface aberta e continua tentando reconectar no plano de fundo de maneira transparente.

3. O FRAMEWORK ANGULAR: A "ESTUFA MODULAR PRÉ-FABRICADA"DA ENGENHARIA

A aplicação frontend do projeto hipotético foi desenvolvida utilizando o **Angular**, um dos mais poderosos frameworks de código aberto criados pela Google para a construção de aplicações web profissionais.

Para um agrônomo, entender o que é o Angular pode ser assimilado através da analogia de uma **estufa agrícola modular pré-fabricada**. Se um produtor decidisse construir uma estufa comprando barras de ferro avulsas, cortando plástico filme manualmente e

projetando cada conexão sem um padrão, a construção levaria meses e seria propensa a falhas estruturais. O Angular atua como o kit de engenharia padronizado: ele fornece a estrutura modular, as normas de organização e os encaixes perfeitos para que a equipe de TI monte a aplicação com rapidez, segurança e alta qualidade.

No código localizado em /sistema/frontend/src/app, a aplicação é organizada em blocos autônomos denominados **Componentes**. Cada componente cuida de uma função específica da irrigação e reúne, em uma única pasta, a sua anatomia (HTML), sua estética (SCSS) e sua inteligência lógica (TypeScript):

- **Componente de Login (*login/*):** Responsável por renderizar a tela de entrada, capturar as credenciais do operador Thiago ou Sérgio e validar o acesso seguro.
- **Componente de Status Atual (*current-status/*):** Exibe os cartões operacionais indicando em tempo real se a motobomba e os quatro setores de irrigação (Melão, Tomate, Pimentão e Estufa) estão ativos ou inativos.
- **Componente de Agendamentos (*irrigation-schedule/*)**: Apresenta a lista de irrigações programadas para o dia, permitindo cadastrar novos horários e cancelar regas antigas.
- **Componente de Agenda Geral (*agenda/)*:** Proporciona a visualização em formato de calendário semanal e mensal do histórico de regas efetuadas na lavoura.

4. ESTRUTURA DO ARQUIVO *index.html* E A HOSPEDAGEM DA APLICAÇÃO WEB

O arquivo index.html (localizado em /sistema/frontend/src/index.html) é a “porta de entrada” absoluta de toda a aplicação web. A listagem a seguir apresenta o conteúdo exato deste arquivo no projeto hipotético.

Listing 2: Arquivo estrutural de entrada index.html da aplicação frontend Angular.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Frontend</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500&display=swap" rel="stylesheet
      ">
  <link  href="https://fonts.googleapis.com/icon?family=Material+Icons"  rel="stylesheet">
</head>
<body class="mat-typography">
  <app-root></app-root>
</body>
</html>
```

Ao analisar a Listagem 1 sob a perspectiva agronômica, destacam-se elementos fundamentais:

1. **Metadados de Responsividade *(<meta name="viewport"...>)*:** Informa ao navegador do celular ou tablet que a tela deve ajustar automaticamente a escala dos elementos visuais de acordo com a largura do dispositivo móvel do operador no campo.
2. **Inclusão de Fontes e Ícones (*Roboto e Material Icons*):** Carrega a tipografia padronizada e a biblioteca de ícones visuais (como gotas de água, motores e indicadores de sinal Wi-Fi).
3. **A Tag de Ancoragem Raiz *(<app-root></app-root>)*:** Esta é a instrução mais importante de todo o arquivo. A tag *<app-root>* funciona como o "vaso de plantio principal" ou ponto de ancoragem onde o Angular insere e remove dinamicamente os componentes visuais da aplicação. Quando o usuário navega do Login para o Dashboard, o código interno de *<app-root>* é substituído sem que a página inteira precise ser recarregada.

4.1. Onde Esses Arquivos Ficam Armazenados na Hospedagem Web?

Uma dúvida frequente de profissionais que não são da área de TI é: "*Onde exatamente ficam guardados esses arquivos HTML, CSS e JavaScript para que o agrônomo possa acessá-los no celular?*".

Durante a fase de desenvolvimento, a equipe de TI escreve código modular. No entanto, para publicar o sistema em produção na internet, é executado um comando de

compilação conhecido como *ng build.* Esse comando processa todo o projeto Angular e gera uma pasta contendo apenas arquivos estáticos hipercompactados (arquivos .html, .js e .css).

Esses arquivos estáticos são transferidos para um **Servidor Web de Hospedagem** (como o Nginx, Apache ou serviços em nuvem como AWS S3 ou Cloudflare). O servidor web de hospedagem funciona como um "depósito de distribuição central": quando o agrônomo digita o endereço https://sergiosouzanovak.com.br/irrigacao no navegador do seu smartphone, o servidor web entrega uma cópia desses arquivos estáticos para o aparelho do usuário. A partir desse instante, a aplicação passa a rodar diretamente dentro da memória do telefone do agrônomo.

5. O PROTOCOLO HTTP: ENTREGA DE PÁGINAS HTML VS. DADOS ESTRUTURADOS JSON

Como explorado no Capítulo 4, o protocolo HTTP é o veículo universal de transporte de dados na internet. No entanto, é crucial compreender que o protocolo HTTP é agnóstico em relação ao tipo de conteúdo que transporta: ele é capaz de entregar tanto **páginas visuais (HTML)** para consumo humano quanto **dados puros (JSON)** para comunicação entre computadores.

A Tabela 1 sintetiza as diferenças fundamentais entre o uso do protocolo HTTP para servir o Frontend e para comunicar dados via API REST.

Tabela 1: Diferenças operacionais nas respostas do protocolo HTTP: entregando o Fron-tend visual vs. dados da API REST.

Característica	HTTP Servindo o Frontend (HTML/JS)	HTTP Servindo a API REST (JSON)
Público-Alvo	Seres humanos (Agrônomos, Operadores)	Programas e Microcontroladores (ESP32, App)
Conteúdo Entregue	Arquivos visuais estáticos (*.html*,*.css*, *.js*)	Strings de dados estruturados em texto JSON
Objetivo	Desenhar a interface gráfica na tela do navegador	Informar valores de umidade e status de válvulas
Frequência de Envio	Baixa (Apenas no primeiro acesso ao sistema)	Alta (Consultas periódicas a cada 5 segundos)
Exemplo de Resposta	<html><body><app-root>...</app-root></body></html>	"bomba":true,"solenoide_1":false

Fonte: Elaborada pelos autores.

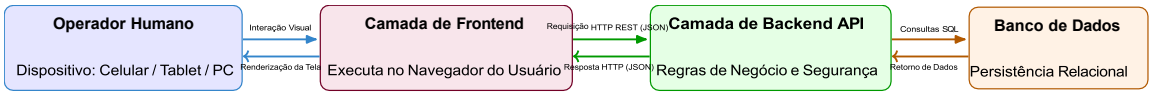
Quando o agrônomo abre o sistema no navegador, o HTTP realiza a entrega de páginas HTML/JS para construir a tela. Uma vez que a tela está desenhada, a própria aplicação em JavaScript passa a utilizar o HTTP em segundo plano para trocar exclusivamente payloads JSON com o servidor central backend.

6. POSICIONAMENTO ARQUITETURAL DO FRONTEND NO SISTEMA GLOBAL

Para visualizar com clareza o papel do Frontend dentro do ecossistema completo de automação agrícola, a Figura 2 apresenta o diagrama topológico de camadas da aplicação hipotética.

Como observado na Figura 2, o Frontend situa-se na extremidade de interação com o ser humano. Ele abstrai completamente a complexidade das consultas SQL do banco de dados e dos comandos elétricos dos relés, oferecendo uma experiência amigável e intuitiva para o usuário.

Figura 19: Posicionamento topológico do Frontend: a interface gráfica executa no dispositivo do usuário e comunica-se exclusivamente com a API Backend via HTTP REST JSON.



7. A REGRA DE OURO DA ENGENHARIA DE SOFTWARE: IMPOSSIBILIDADE E PROIBIÇÃO DE CONEXÃO DIRETA FRONTEND-BANCO DE DADOS

Um dos conceitos mais críticos que todo gestor agrícola e profissional de tecnologia deve assimilar é a **Regra de Ouro da Arquitetura Web: O Frontend NUNCA deve, em hipótese alguma, conectar-se diretamente ao Banco de Dados Relacional (MySQL).**

É bastante comum que pessoas leigas se questionem: *"Se o objetivo da tela é exibir os agendamentos salvos no banco de dados, por que a aplicação Angular no celular não abre uma conexão direta com o MySQL e busca os dados de uma vez, sem precisar do backend?*".

Existem três razões fundamentais — de segurança, arquitetura e integridade — que proíbem severamente essa prática:

7.1. Motivo de Segurança Máxima e Exposição de Credenciais

O código da aplicação Frontend (HTML, CSS e JavaScript) é entregue integralmente ao navegador do usuário para ser executado dentro do celular ou computador do cliente. Isso significa que qualquer pessoa mal-intencionada pode abrir as "Ferramentas do Desenvolvedor" do navegador e inspecionar todo o código-fonte da aplicação.

Se o Frontend se conectasse diretamente ao banco de dados MySQL, o código em JavaScript precisaria conter a senha de acesso ao banco (*db_user, db_password*) e o endereço IP do servidor. Um invasor na internet poderia extrair essas credenciais em segundos, acessar o banco de dados da fazenda e apagar todos os registros operacionais, alterar senhas de acesso ou sequestrar os dados da propriedade rural.

Ao interpor a camada de **Backend API** numa arquitetura em camadas (*n-tier*), as credenciais do banco de dados ficam isoladas em um servidor protegido (SOMMER-

VILLE, 2016; BASS; CLEMENTS; KAZMAN, 2012). O Frontend no celular nunca

armazena a senha do banco de dados: ele apenas envia um token de segurança (JWT) para a API Flask, que válida a autorização e as *claims* contidas na assinatura antes de realizar qualquer transação (MARTIN, 2017; OWASP FOUNDATION, 2024).

7.2. Restrições de Protocolos de Rede e Sandbox do Navegador

Navegadores web modernos (como Google Chrome, Apple Safari e Mozilla Firefox) operam sob um rígido modelo de segurança isolado denominado *Sandbox*. Por normas internacionais de segurança da Web, o navegador só tem permissão para abrir conexões utilizando os protocolos HTTP, HTTPS e WebSockets.

O banco de dados MySQL utiliza um protocolo binário proprietário de baixo nível operando na porta TCP 3306. Os navegadores web simplesmente não possuem capacidade nativa para abrir e gerenciar soquetes binários diretos de banco de dados. O backend Flask atua como o tradutor universal: ele recebe a requisição amigável em HTTP/HTTPS vinda do navegador e a converte em consultas SQL nativas para o MySQL.

7.3. Garantia e Validação das Regras de Negócio Agronômicas

Se o Frontend tivesse acesso direto para escrever dados no MySQL, nada impediria que um usuário acidentalmente inserisse um agendamento conflitante ou com parâmetros absurdos (como agendar uma irrigação com duração de 48 horas seguidas, queimando o motor da bomba).

O Backend funciona como um **Guardião de Integridade**. Antes de permitir que qualquer dado seja gravado no MySQL, o servidor Flask revalida todas as regras de segurança:

- Verifica se o tempo de irrigação solicitado respeita o limite máximo de 30 minutos por setor;
- Checa se há colisão de horários entre o setor do melão e o setor do tomate (impedindo a queda de pressão hidráulica);
- Confirma se o usuário possui nível de permissão de agrônomo responsável para autorizar a rega.

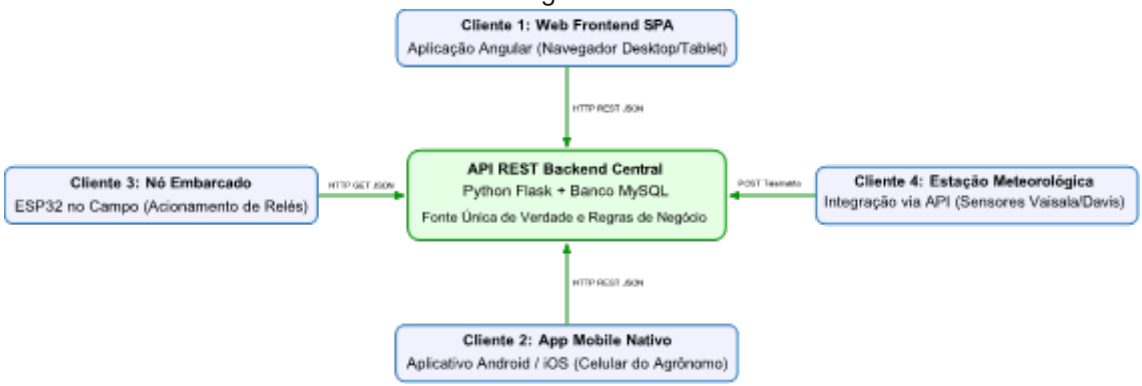
8. O DESACOPLAMENTO FRONTEND-BACKEND E A FACILIDADE DE INTEGRAÇÃO MULTICLIENTE

A separação estrita entre a camada de apresentação (Frontend Angular) e a camada de inteligência e dados (Backend Flask) é chamada na engenharia de software de **Arquitetura Desacoplada** (*Decoupled Architecture*).

A grande vantagem estratégica dessa abordagem para a agricultura digital é que a API Backend Flask torna-se uma **fonte única e universal de serviços**. Uma vez que a API backend está desenvolvida e rodando no servidor central, a empresa agrícola pode criar múltiplos clientes diferentes para consumir os mesmos dados, sem precisar alterar uma única linha de código no backend.

A Figura 3 ilustra o ecossistema multicliente viabilizado pelo desacoplamento da API REST Backend. Como demonstrado na Figura 3, o desacoplamento proporciona expansibilidade ilimitada para o sistema de irrigação:

Figura 20: Ecossistema multicliente viabilizado pelo desacoplamento: o mesmo backend central atende simultaneamente o Frontend Web, aplicativos móveis, microcontroladores ESP32 e estações meteorológicas externas.



1. **Aplicativos Móveis Nativos (Android / iOS):** Caso a fazenda deseje desenvolver um aplicativo nativo para smartphones em Flutter ou React Native, os desenvolvedores de aplicativo utilizarão exatamente os mesmos endpoints REST (*GET/estado-geral, POST /agendamentos*) já criados para o Frontend Web.
2. **Dispositivos Embarcados de Campo (ESP32 / Arduino):** Conforme demonstrado no Capítulo 5, o microcontrolador ESP32 consome a mesma API REST que o painel web utiliza, garantindo total sincronia entre o campo e a gestão.
3. **Integração com Sistemas Externos e Estações Climatológicas:** O sistema pode receber automaticamente dados de precipitação pluviométrica e umidade relativa do ar vindos de estações meteorológicas automáticas parceiras, enviando telemetria via chamadas *POST/telemetria* para alimentar os algoritmos de balanço hídrico.

9. A NECESSIDADE DA PROGRAMAÇÃO ORIENTADA A OBJETOS (POO) EM SISTEMAS WEB AGRÍCOLAS

À medida que o software de automação agrícola cresce para gerenciar múltiplos talhões, diversos tipos de sensores e atuadores, a escrita de código em formato puramente procedural torna-se insustentável. O código transforma-se no que a engenharia de software clássica classifica como “código espaguete”— uma massa emaranhada de linhas com alto acoplamento, onde qualquer alteração provoca erros imprevisíveis no sistema (PRESSMAN; MAXIM, 2020). Para contornar essa degradação, adotam-se princípios de arquitetura limpa e padrões de projeto que favorecem a modularidade e a manutenibilidade (GAMMA et al., 1994).

Para resolver esse problema e garantir escalabilidade, o framework Angular e a linguagem TypeScript adotam estritamente o paradigma da **Programação Orientada a Objetos (POO).**

Para compreender a estrutura interna do frontend, os pilares da POO devem ser analisados sob a ótica da engenharia de software aplicada ao framework Angular e à linguagem TypeScript:

9.1. Classe (*Class*) vs. Objeto (*Object/Instance*)

A distinção entre Classe e Objeto é o conceito estrutural primário da Programação Orientada a Objetos aplicada ao desenvolvimento Web:

- **Classe (Class) no Angular:** Uma Classe em TypeScript é a definição estática em tempo de compilação (*compile-time*) da estrutura de um componente visual, serviço ou modelo de dados (por exemplo, *CurrentStatusComponent* ou *AuthService*). A Classe declara os atributos da estrutura (como o estado de autenticação ou a lista de solenoides) e os métodos de manipulação de lógica (como *login(), logout()* e *toggleSolenoide()*). A Classe funciona como a especificação de um tipo de dados: ela especifica as assinaturas de métodos e propriedades, mas não armazena estados ativos em memória de execução até ser instanciada.
- **Objeto (*Object/Instance*) no Runtime:** Um Objeto é a alocação dinâmica e concreta de memória RAM realizada no navegador durante o tempo de execução (*run-time*). Quando a aplicação Angular executa no browser e o usuário navega para uma determinada visão, o container de injeção do Angular invoca o construtor da classe e cria o Objeto correspondente na memória do cliente, vinculando seu estado ao ciclo de vida do componente (através dos ganchos *ngOnInit()* e *ngOnDestroy()*). De maneira análoga, ao receber o payload JSON da API backend contendo os estados dos atuadores, o Angular instancia Objetos de modelo em memória, permitindo que cada solenoide (Setor 1 Melão, Setor 2 Tomate, etc.) tenha seus atributos dinâmicos manipulados individualmente em tempo real.

9.2. Encapsulamento (Encapsulation)

O Encapsulamento é a prática de proteger o estado interno de um objeto contra modificações indevidas ou não autorizadas por código externo. Em termos agronômicos, é como a casca protetora de uma semente ou a epiderme impermeável de um fruto, protegendo os tecidos internos contra contaminantes externos.

No arquivo *auth.service.ts* da aplicação em /sistema/frontend/src/app/auth.servi ce.ts, o estado de autenticação do usuário é rigorosamente encapsulado:

```
@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private loggedIn = new BehaviorSubject<boolean>(false);
  private currentUser = new BehaviorSubject<string | null>(null);

  login(username: string, password: string): boolean { ... }
  isLoggedIn(): Observable<boolean> { return this.loggedIn.asObservable(); }
}
```

Ao definir o atributo como private loggedIn, o programador garante que nenhum outro componente visual possa alterar arbitrariamente a variável de acesso para true sem

passar pelo método oficial login(), onde a senha e o nome do operador são devidamente auditados.

9.3. Abstração e Injeção de Dependência (Dependency Injection)

A Abstração permite isolar a complexidade interna de um serviço por trás de uma interface simples de utilizar. O agrônomo que opera o painel web não precisa saber qual equação diferencial o backend executa ou qual chave criptográfica TLS é negociada: ele apenas clica no botão e o sistema abstrai toda a complexidade.

No Angular, a **Injeção de Dependência** é a técnica orientada a objetos onde os componentes gráficos não criam manualmente seus serviços de dados (usando o operador new). Em vez disso, o container do Angular "injeta" automaticamente as instâncias de serviços compartilhados (como o *AuthService* e o *IrrigationService*) diretamente no construtor da classe do componente. Isso garante que todos os componentes da aplicação consultem exatamente a mesma fonte de dados, eliminando discrepâncias no painel.

10. MODELAGEM DO FRAMEWORK ANGULAR VIA DIAGRAMA DE COMPONENTES UML

Para formalizar a arquitetura de software da aplicação frontend do projeto hipotético, utiliza-se o **Diagrama de Componentes** UML (*Unified Modeling Language*, versão 2.5).

Em engenharia de software, o Diagrama de Componentes descreve como os módulos lógicos (componentes, serviços, roteadores e coleções de dados) estão organizados e como eles se interconectam através de interfaces providas e requeridas.

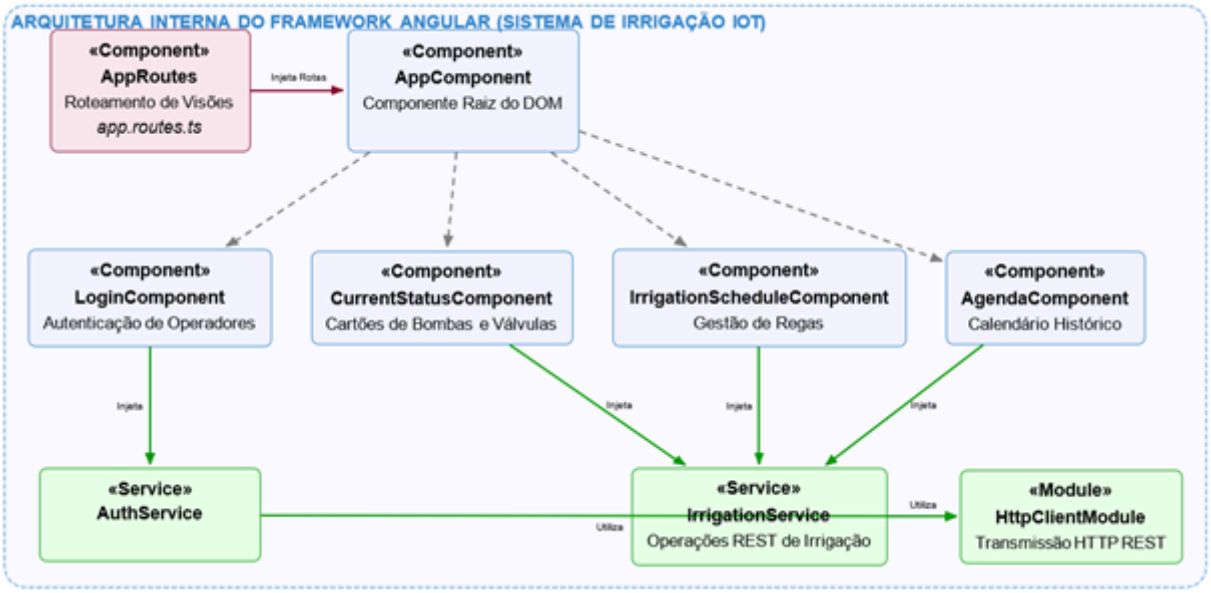
A Figura 4 apresenta o Diagrama de Componentes UML completo da aplicação localizada em /sistema/frontend/src/app.

10.1. Análise Detalhada dos Elementos do Diagrama UML

A análise da arquitetura representada no Diagrama de Componentes da Figura 4 revela como as responsabilidades estão distribuídas:

1 **Componente de Roteamento (AppRoutes): Atua como o "despachante** central"de caminhos da aplicação. Quando o usuário digita a URL ou clica em um botão do menu, o AppRoutes identifica a rota (ex: /login ou /status) e carrega o componente correspondente no container raiz.

Figura 21: Diagrama de Componentes UML do framework Angular: componentes de interface gráfica injetam serviços especializados e comunicam-se via HttpClientModule.



2 **Componentes Visuais de Interface (Login, CurrentStatus, Schedule, Agenda):** São os blocos de interação com o usuário. Eles não contêm lógica de conexão de rede embutida: limitam-se a capturar eventos da tela e renderizar os dados recebidos dos serviços.

3 **Serviços Especializados (*AuthService* e *IrrigationService*):** São classes orienta-das a objetos decoradas com a anotação @I*njectable*(). Eles concentram a lógica de chamadas REST, mantêm o estado atual da sessão e notificam os componentes visuais sobre atualizações através de fluxos reativos (*Reactive Streams* com *Beha-viorSubject* e *Observable*).

4 **Módulo de Comunicação HTTP (*HttpClientModule*):** É o componente nativo do Angular responsável por empacotar as chamadas REST em formato JSON e enviá-las através da rede sem fio para o servidor central Flask.

11. SÍNTESE DO CAPÍTULO E CONCLUSÃO GERAL DA OBRA

Este capítulo apresentou os conceitos fundamentais do Frontend Web aplicados à automação de irrigação agrícola, traduzindo termos complexos de tecnologia para uma linguagem acessível a engenheiros agrônomos e profissionais do campo.

Ao analisar a analogia com a estrutura vegetal (HTML como esqueleto morfológico,

CSS como estética e corte ceroso, e JavaScript como fisiologia e movimento de seiva), compreende-se com clareza o papel de cada linguagem na construção de interfaces gráficas. A adoção da arquitetura **SPA (*Single Page Application*)** com o framework Angular comprovou-se a escolha ideal para cenários agrícolas rurais, proporcionando navegação fluida sem recarregamentos de tela e otimizando o consumo de dados em conexões sem fio.

Ficou demonstrada a importância da R**egra de Ouro da Arquitetura:** a proibição da conexão direta entre o Frontend e o Banco de Dados MySQL por motivos de segurança das credenciais, limites de protocolos de rede e validação rigorosa das salvaguardas de irrigação pelo Backend. Além disso, foram apresentados os princípios da **Programação Orientada a Objetos (POO)** e a modelagem formal da aplicação Angular por meio do **Diagrama de Componentes UML**, ressaltando a modularidade, o encapsulamento e a injeção de dependências.

Com a conclusão deste capítulo, a obra completa um panorama abrangente, rigoroso e altamente prático sobre a engenharia de sistemas de irrigação automatizados com Internet das Coisas:

- **Capítulo 1:** Introdução às interfaces web e ecossistema de irrigação IoT;
- **Capítulo 2:** Especificações dos equipamentos, sensores capacitivos, válvulas solenoides e motobombas;
- **Capítulo 3:** Circuitos eletreletrônicos de amostragem, optoacopladores e chaves contatoras de potência;
- **Capítulo 4:** Fundamentos do protocolo HTTP REST e arquitetura cliente-servidor;
- **Capítulo 5:** Firmware C++ em microcontrolador ESP32, modelo de Cliente Magro (*Thin Client*) de Tanenbaum e salvaguardas falha-segura;
- **Capítulo 6:** Arquitetura Frontend SPA Angular, Programação Orientada a Objetos, Diagrama de Componentes UML, desacoplamento de backend e ecossistemas multicliente para a lavoura digital.

REFERÊNCIAS

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software Architecture in Practice**. 3. ed. Boston: Addison-Wesley, 2012.

GAMMA, Erich et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. Reading: Addison-Wesley, 1994.

MARTIN, Robert C. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. Boston: Prentice Hall, 2017.

OWASP FOUNDATION. **REST Security Cheat Sheet**. [S.l.: s.n.], 2024. Disponível em: <https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html>.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Software Engineering: A Practitioner's Approach**. 9. ed. New York: McGraw-Hill, 2020.

SOMMERVILLE, Ian. **Software Engineering**. 10. ed. Boston: Pearson, 2016.

CAPÍTULO 7

BACKEND PYTHON FLASK: API RESTFUL, ARQUITETURA DE ROTAS, PERSISTÊNCIA MYSQL E TESTES DE INTEGRAÇÃO NA IRRIGAÇÃO IoT

1. INTRODUÇÃO AO BACKEND PARA O LEITOR AGRÔNOMO: O MOTOR CENTRAL DA FAZENDA DIGITAL

Para um engenheiro agrônomo, produtor rural ou gestor de lavoura, a compreensão da infraestrutura computacional que sustenta a agricultura digital torna-se clara ao utilizar a analogia de uma **Casa de Máquinas Central** em um projeto de irrigação por gotejamento ou pivô central.

Em uma fazenda automatizada, a lavoura pode possuir quilômetros de tubulações hidráulicas, dezenas de válvulas solenoides instaladas nos talhões e painéis solares no campo. No entanto, nenhum setor irrigará uma única gota de água se a casa de máquinas central — onde ficam alocados a motobomba principal, os dosadores de fertirrigação, os filtros de disco, os quadros elétricos de comando e os manômetros de pressão — não estiver funcionando perfeitamente e enviando o fluxo de água correto para o setor desejado.

Na engenharia de software, o **Backend** é exatamente a "casa de máquinas central" do sistema de informação (SOMMERVILLE, 2016). Enquanto o *Frontend* (Capítulo 6) representa o painel visual com o qual o operador humano interage e os microcontroladores ESP32 (Capítulo 5) representam os acionadores elétricos no campo, o Backend é a inteligência invisível que roda em um servidor remoto, processando as regras de negócio, gerenciando as conexões com o banco de dados e garantindo que o comando enviado pelo agrônomo seja executado com total segurança (MARTIN, 2017).

Na aplicação hipotética desenvolvida no diretório /sistema/backend-irriga, o servidor Backend foi construído utilizando a linguagem de programação **Python** e o framework web **Flask**. Este capítulo tem por objetivo explicar detalhadamente cada componente desse backend, apresentando seus códigos-fonte, sua arquitetura de rotas, seu banco de dados MySQL e as ferramentas de teste de integração.

2. O FRAMEWORK PYTHON FLASK: O CHASSI ÁGIL DA AUTOMAÇÃO

No ecossistema de desenvolvimento de software em linguagem Python, existem diversos frameworks para a criação de aplicações web. O **Flask** é classificado pela comunidade técnica como um *micro-framework*.

Para o leitor agrônomo, o conceito de *micro-framework* pode ser compreendido mediante a comparação entre dois tipos de maquinários agrícolas:

- **Frameworks Pesados (Estilo Colheitadeira Rígida):** Frameworks como o Django fornecem uma estrutura gigante e pré-configurada "tudo-em-um". São excelentes para grandes portais corporativos, mas trazem um peso computacional elevado e pouca flexibilidade para customizações de baixo nível necessárias na

Internet das Coisas.

- **Micro-Framework Flask (Estilo Trator Utilitário Modular):** O Flask fornece apenas o essencial — o chassi mecânico ágil com motor de alto rendimento, o sistema de roteamento de mensagens HTTP e o gerenciamento de requisições. Cabe aos desenvolvedores acoplar apenas os implementos estritamente necessários (como a biblioteca SQLAlchemy para banco de dados e o repositório de rotas Blueprints).

Devido à sua extrema leveza, rapidez de resposta e eficiência no processamento de requisições concorrentes, o Flask é a escolha ideal para atuar como o cérebro central de plataformas de irrigação inteligente IoT.

3. CONCEITUAÇÃO DE ROTAS HTTP E ENDPOINTS USANDO O DOMÍNIO www.meusitedeirrigacao.com

Para que a interface web em Angular, o aplicativo de celular do agrônomo ou o micro-controlador ESP32 consigam conversar com o servidor Flask, eles utilizam **Rotas HTTP** (também chamadas de *Endpoints*).

Uma Rota HTTP funciona como um **endereço postal digital** ou uma **válvula seletora de distribuição hidráulica**. Cada rota possui um nome de caminho específico e aceita um ou mais métodos HTTP (conforme apresentado no Capítulo 4), definindo exatamente qual ação o servidor deve executar.

Para ilustrar o funcionamento prático das rotas no sistema hipotético, consideremos que a plataforma de irrigação da fazenda está publicada na internet sob o domínio corporativo oficial:

http://www.meusitedeirrigacao.com

A seguir, apresentam-se exemplos práticos de como cada método HTTP é utilizado no domínio www.meusitedeirrigacao.com para controlar a infraestrutura de irrigação.

3.1. Método GET: Consulta e Leitura de Dados

O método *GET* é utilizado quando um cliente (navegador ou ESP32) deseja ler informações armazenadas no servidor, sem alterar nenhuma configuração no banco de dados:

- **Consulta do Estado Geral pelo ESP32:**

GET http://www.meusitedeirrigacao.com/estado-geral

O servidor processa a consulta, verifica os agendamentos ativos para o horário atual

e retorna o documento JSON informando se a bomba e as quatro solenoides devem estar ligadas.

- **Listagem de Agendamentos pelo Frontend Web:**

GET http://www.meusitedeirrigacao.com/agendamentos

O servidor realiza uma busca na tabela MySQL e devolve a lista completa de todas as regas programadas na fazenda.

3.2. Método POST: Criação e Disparo de Comandos

O método *POST* é utilizado para submeter novos dados ao servidor, criando registros no banco de dados ou alterando manualmente o estado elétrico de um atuador:

- **Cadastro de Novo Agendamento de Irrigação:**

POST http://www.meusitedeirrigacao.com/agendamentos

O formulário web do agrônomo envia no corpo da requisição os dados da nova rega (data, horário de início, duração e solenoide selecionada). O servidor Flask valida as informações e insere uma nova linha na tabela do MySQL.

- **Acionamento Manual de Solenoide de Campo:**

POST http://www.meusitedeirrigacao.com/solenoide?id=1&ativo=true

O operador força a abertura imediata da solenoide 1 (Setor Melão), atualizando o banco de dados.

3.3. Método PUT: Atualização Completa de Registros

O método *PUT* é empregado para modificar as informações de um recurso já existente no sistema:

- **Alteração de Horário de um Agendamento Existente:**

PUT http://www.meusitedeirrigacao.com/agendamentos/12

O agrônomo decide prorrogar a rega do agendamento de identificador ID 12. O

cliente envia os novos horários e o servidor executa uma instrução SQL do tipo *UPDATE* na linha correspondente do banco MySQL.

3.4. Método DELETE: Remoção e Cancelamento

O método *DELETE* é utilizado para excluir permanentemente um registro do banco de dados:

- **Cancelamento de uma Rega Programada:**

DELETE http://www.meusitedeirrigacao.com/agendamentos/12

O operador cancela a rega ID 12 porque começou a chover na propriedade. O servidor recebe a instrução e executa o comando *SQL DELETE FROM* agendamentos *WHERE id = 12.*

4. ANÁLISE ESTRUTURAL E CÓDIGO-FONTE DO ARQUIVO PRINCIPAL *app.py*

O arquivo *app.py* (localizado em /sistema/backend-irriga/app.py) é o ponto de entrada principal do servidor Flask. Ele é responsável por inicializar a aplicação web, configurar a conexão com o banco de dados MySQL, registrar os módulos de rotas e iniciar o servidor de escuta.

A Listagem 1 apresenta o código-fonte exato do arquivo *app.py.*

Listing 4: Código-fonte do arquivo principal app.py do servidor backend Flask.

```
import os
import sys
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import text
from flask_cors import CORS
from routes.bomba_routes import bomba_routes
from routes.agendamento_routes import agendamento_routes
import urllib.parse

from db import db
app = Flask(__name__)
app.secret_key = 'sua_chave_secreta_aqui'

CORS(app, expose_headers=["X-Filename", "Content-Disposition"])

password = "Irriga@2026"
encoded_password = urllib.parse.quote_plus(password)

db_host = os.environ.get('DB_HOST', 'localhost')
app.config['SQLALCHEMY_DATABASE_URI'] = (
    f'mysql+pymysql://sergiosouzanovak_sergiosouzanovak:{encoded_password}'
    f'@{db_host}/sergiosouzanovak_sistema_irrigacao'
)
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SQLALCHEMY_ENGINE_OPTIONS'] = {
    'pool_recycle': 280,
    'pool_pre_ping': True,
}
db.init_app(app)
app.register_blueprint(bomba_routes)
app.register_blueprint(agendamento_routes)

if __name__ == "__main__":
    app.run(debug=True, host='0.0.0.0', port=5000)
```

4.1. Análise Linha a Linha das Configurações do Servidor

A análise técnica do arquivo *app.py* revela decisões cruciais de arquitetura de software:

1. **Importação de Módulos e Bibliotecas (Linhas 1 a 12):** O script importa as classes do Flask, a extensão *SQLAlchemy* para mapeamento de banco de dados, o manipulador de segurança de rede *CORS* e os dois conjuntos de rotas contidos na pasta *routes/.*
2. **Tratamento de Senha de Banco de Dados com Caracteres Especiais (Linhas 19 e 20):** A senha do banco de dados MySQL contém o caractere especial @ *("Irriga@2026")*. Como a URL de conexão com o banco utiliza o caractere @ para separar a senha do endereço IP do servidor, o código utiliza a função *urllib.parse.quote_plus()* para codificar o caractere em formato percentual *(%40*). Isso impede que o Python confunda a senha com o endereço de rede do banco MySQL.
3. **Configuração da Conexão MySQL (Linhas 23 a 26):** A variável *SQLALCHEMY_DATABASE_URI* monta a string de conexão utilizando o driver *pymysql*, conectando-se ao banco de dados relacional *sergiosouzanovak_sistema_irrigacao.*
4. **Gestão de Pool de Conexões e Reconexão Autônoma (Linhas 28 a 31):** Em servidores de hospedagem web rurais, as conexões inativas com o banco de dados MySQL podem ser encerradas por limite de tempo (*timeout*). A configuração *'pool_recycle': 280* força o Flask a renovar as conexões a cada 280 segundos. Além disso, o parâmetro *'pool_pre_ping*': True ativa a sondagem prévia: antes de exe-cutar qualquer consulta SQL, o Flask testa a conexão; se o MySQL tiver caído, o Flask reconecta automaticamente de forma transparente.
5. **Registros de Blueprints de Rotas (Linhas 33 e 34):** A instrução *app.register_blueprint()* acopla os módulos independentes de rotas de atuadores (*bomba_routes*) e de agendamentos (*agendamento_routes*) à aplicação principal, garantindo uma estrutura modular de código.

5. A LÓGICA DE SINCRONIZAÇÃO AUTOMÁTICA DE STATUS: A FUNÇÃO *sincronizar_status()*

Um dos pontos mais geniais e importantes de todo o backend está localizado no arquivo *bomba_routes.py:* a função *sincronizar_status()* (linhas 8 a 54 da Listagem 2).

Para o leitor agrônomo, entender essa função é compreender a **automação em tempo real**. O servidor não depende que uma pessoa clique em um botão no momento exato em que a irrigação deve começar. A função *sincronizar_status()* atua como um relógio automático inteligente que avalia a hora atual do dia contra a tabela de agendamentos salvos.

5.1. O Algoritmo de Decisão de Acionamento Hidráulico

Sempre que o ESP32 ou o painel web realizam uma consulta ao servidor, a função *sincronizar_status()* executa as seguintes etapas:

1. **Leitura de Todos os Solenoides Cadastrados:** O servidor consulta a tabela MySQL (*SELECT id, nome, ativo FROM solenoides).*
2. **Avaliação Temporal contra Agendamentos Ativos:** Para cada solenoide (Melão, Tomate, Pimentão, Estufa), o servidor executa uma consulta SQL especial que testa se a hora atual do relógio do servidor (*CURTIME())* e a data de hoje (*CURDATE()*) estão dentro da janela de irrigação cadastrada pelo agrônomo:

```
SELECT COUNT(*) FROM agendamentos
WHERE data_agendamento = CURDATE()
  AND CURTIME() BETWEEN hora_inicio AND hora_fim
  AND solenoide = :nome
```

3. **Atualização Dinâmica do Estado do Solenoide:** Se a contagem retornar um valor maior que zero *(has_schedule > 0*), significa que aquele setor **deve estar irrigando neste exato minuto.** Se o banco de dados indicava que ele estava desligado, o servidor executa o comando SQL UPDATE solenoides *SET ativo = 1 WHERE id= :id*. Caso contrário, se o horário de irrigação já passou, o servidor atualiza para *ativo = 0.*
4. **Decisão Inteligente de Acionamento da Motobomba Principal:** O algoritmo avalia se existe qualquer solenoide ativo na fazenda ou qualquer agendamento em execução. Se houver pelo menos um setor precisando de água, a motobomba principal de 5,5 CV é automaticamente marcada para ligar (*UPDATE bomba SET ativo = 1*). Se todos os setores estiverem inativos, a motobomba é desligada (*UPDATE bomba SET ativo = 0*), protegendo a bomba contra funcionamento sem vazão e economizando energia elétrica.

Essa função garante que, mesmo que o celular do agrônomo esteja desligado ou sem bateria, a fazenda continuará sendo irrigada exatamente nos horários agendados.

6. ANÁLISE DETALHADA DAS ROTAS DE ATUADORES (bomba_routes.py)

O *arquivo bomba_routes.py* (localizado em /sistema/backend-irriga/routes/bomba_ rou tes.py) agrupa todas as rotas responsáveis pelo controle dos equipamentos físicos (moto-bomba e válvulas solenoides).

A Listagem 2 apresenta a implementação completa do módulo de rotas de atuadores.

Listing 5: Código-fonte das rotas de controle de bomba e solenoides (bomba_routes.py).

```
from flask import Blueprint, jsonify, request
from sqlalchemy import text
from db import db



bomba_routes = Blueprint('bomba_routes', __name__)



def sincronizar_status():
    solenoides = db.session.execute(text("SELECT id, nome, ativo FROM solenoides")).fetchall()



    any_solenoide_ativo = False
    for s_id, s_nome, s_ativo in solenoides:
        sql_check = """
        SELECT COUNT(*) FROM agendamentos
        WHERE data_agendamento = CURDATE()
          AND CURTIME() BETWEEN hora_inicio AND hora_fim AND
          solenoide = :nome
        """
        has_schedule = db.session.execute(text(sql_check), {"nome": s_nome}).scalar() or 0
        deve_estar_ativo = has_schedule > 0

        if deve_estar_ativo != bool(s_ativo):
            db.session.execute(
```

```
                    text("UPDATE solenoides SET ativo = :ativo WHERE id = :id"),
                    {"ativo": 1 if deve_estar_ativo else 0, "id": s_id}

                )


            if deve_estar_ativo:
                any_solenoide_ativo = True

        bomba = db.session.execute(text("SELECT id, ativo FROM bomba LIMIT 1")).fetchone()
        if bomba:
            id_bomba, bomba_ativo = bomba[0], bool(bomba[1])

            sql_agendamentos = """
            SELECT COUNT(*) FROM agendamentos
            WHERE data_agendamento = CURDATE()
              AND CURTIME() BETWEEN hora_inicio AND hora_fim
            """
            agendamentos_ativos = db.session.execute(text(sql_agendamentos)).scalar() or 0

            deve_ligar_bomba = (agendamentos_ativos > 0) or any_solenoide_ativo

            if deve_ligar_bomba != bomba_ativo:
                db.session.execute(
                    text("UPDATE bomba SET ativo = :ativo WHERE id = :id"),
                    {"ativo": 1 if deve_ligar_bomba else 0, "id": id_bomba}
                )

        db.session.commit()

@bomba_routes.route('/bomba', methods=['GET'])
def get_bomba():
    try:
        sincronizar_status()
        sql = "SELECT id, nome, ativo FROM bomba LIMIT 1"
        row = db.session.execute(text(sql)).fetchone()
        if row:
            return jsonify({"id": row[0], "nome": row[1], "ativo": bool(row[2])}), 200
        else:
            return jsonify({"error": "Bomba nao encontrada"}), 404
    except Exception as e:
        print(e)
        return jsonify({"error": "Erro interno"}), 500
```

```
@bomba_routes.route('/bomba', methods=['POST'])
def set_bomba():
    try:
        ativo = request.args.get('ativo')
        if ativo is None:
            return jsonify({"error": "Parametro 'ativo' obrigatorio"}), 400
        ativo_bool = ativo.lower() == 'true'
        sql = "UPDATE bomba SET ativo = :ativo WHERE id = 1"
        db.session.execute(text(sql), {"ativo": ativo_bool})
        db.session.commit()
        return jsonify({"message": "Bomba atualizada", "ativo": ativo_bool}), 200
    except Exception as e:
        print(e)
        return jsonify({"error": "Erro interno"}), 500

@bomba_routes.route('/solenoides', methods=['GET'])
def get_solenoides():
    try:
        sql = "SELECT id, nome, ativo FROM solenoides"
        rows = db.session.execute(text(sql)).fetchall()
        resultado = [{"id": row[0], "nome": row[1], "ativo": bool(row[2])} for row in rows]
        return jsonify(resultado), 200
    except Exception as e:
        print(e)
        return jsonify({"error": "Erro interno"}), 500

@bomba_routes.route('/solenoide', methods=['POST'])
def set_solenoide():
    try:
        id_solenoide = request.args.get('id')
        ativo = request.args.get('ativo')
        if id_solenoide is None or ativo is None:
            return jsonify({"error": "Parametros 'id' e 'ativo' sao obrigatorios"}), 400
        ativo_bool = ativo.lower() == 'true'
        sql = "UPDATE solenoides SET ativo = :ativo WHERE id = :id"
        db.session.execute(text(sql), {"ativo": ativo_bool, "id": id_solenoide})
        db.session.commit()
        return jsonify({"message": f"Solenoide {id_solenoide} atualizado", "ativo": ativo_bool}), 200
    except Exception as e:
        print(e)
        return jsonify({"error": "Erro interno"}), 500

```

```
@bomba_routes.route('/estado-geral', methods=['GET'])
def estado_geral():
    try:
        sincronizar_status()
        solenoides = db.session.execute(text("SELECT id, ativo FROM solenoides")).fetchall() bomba
        = db.session.execute(text("SELECT ativo FROM bomba WHERE id = 1")).scalar() return
        jsonify({
            "bomba": bool(bomba),
            "solenoides": [{"id": s[0], "ativo": bool(s[1])} for s in solenoides]
        }), 200
    except Exception as e:
```

6.1. O Endpoint */estado-geral*: A Rota de Consumo dos Microcontroladores

A rota mais importante para o funcionamento da Internet das Coisas na lavoura é o end-point */estado-geral* (linhas 150 a 174 da Listagem 2).

Quando o cliente magro ESP32 (Capítulo 5) dispara uma requisição *GET http://www.meusitedeirrigacao.com/estado-geral*, o servidor Flask aciona primeiro a função *sincronizar_status(),* garantindo que o banco de dados reflita o horário exato. Em seguida, o servidor lê o estado da bomba e das quatro solenoides e retorna a resposta JSON compacta:

```
{
  "bomba": true,
  "solenoides": [
    { "id": 1, "ativo": true },
    { "id": 2, "ativo": false },
    { "id": 3, "ativo": false },
    { "id": 4, "ativo": false }
  ]
}
```

Esse payload JSON consome menos de 150 bytes de banda de rede, permitindo que o ESP32 interprete os dados em milissegundos mesmo em redes Wi-Fi rurais instáveis.

7. ANÁLISE DETALHADA DAS ROTAS DE AGENDAMENTO (agendamento_routes.py)

O arquivo *agendamento_routes.py* (localizado em /sistema/backend-irriga/routes/agend amento_routes.py) gerencia todas as operações de criação, leitura, atualização e deleção (*CRUD*) das rotinas de rega cadastradas pelos operadores humanos.

A Listagem 3 apresenta o código-fonte completo das rotas de agendamento.

Listing 6: Código-fonte completo das rotas de gestão de agendamentos (agendamento_routes.py).

```
import datetime
from flask import Blueprint, jsonify, request
from sqlalchemy import text
from db import db



agendamento_routes = Blueprint('agendamento_routes', __name__)



def format_row(row):
    id, data_agendamento, hora_inicio, hora_fim, duracao_minutos, solenoide, usuario, status = row
    date_str = data_agendamento.isoformat() if hasattr(data_agendamento, 'isoformat') else str(data_agendamento)



    def format_time(t):
        if hasattr(t, 'strftime'):
            return t.strftime('%H:%M:%S')
        if isinstance(t,
            datetime.timedelta): seconds
            = t.total_seconds() hours =
            int(seconds // 3600)

            minutes = int((seconds % 3600) // 60)
            secs = int(seconds % 60)

            return f"{hours:02d}:{minutes:02d}:{secs:02d}"
        return str(t)
```

```
    }

@agendamento_routes.route('/agendamentos', methods=['GET'])
def get_agendamentos():
    try:
        sql = """
        SELECT id, data_agendamento, hora_inicio, hora_fim, duracao_minutos, solenoide, usuario, status
        FROM agendamentos ORDER BY data_agendamento DESC, hora_inicio DESC
        """
        rows = db.session.execute(text(sql)).fetchall()
        result = [format_row(row) for row in rows]
        return jsonify(result), 200
    except Exception as e:
        print(e)
        return jsonify({"error": f"Erro interno: {str(e)}"}), 500

@agendamento_routes.route('/agendamentos/<int:id>', methods=['GET'])
def get_agendamento(id):
    try:
        sql = "SELECT id, data_agendamento, hora_inicio, hora_fim, duracao_minutos, solenoide, usuario, status
            FROM agendamentos WHERE id = :id"
        row = db.session.execute(text(sql), {"id": id}).fetchone()
        if row:
            return jsonify(format_row(row)), 200
        return jsonify({"error": "Agendamento nao encontrado"}), 404
    except Exception as e:
        print(e)
        return jsonify({"error": f"Erro interno: {str(e)}"}), 500

@agendamento_routes.route('/agendamentos', methods=['POST'])
def create_agendamento():
    try:
        data = request.get_json(silent=True) or {}
        if not data:
            data = request.form.to_dict() or request.args.to_dict()

        data_agendamento = data.get('data_agendamento') or data.get('date')
        hora_inicio = data.get('hora_inicio') or data.get('startTime')
        hora_fim = data.get('hora_fim') or data.get('endTime')
        duracao_minutos = data.get('duracao_minutos') or data.get('durationMinutes')
        solenoide = data.get('solenoide') or data.get('solenoid')
        usuario = data.get('usuario') or data.get('scheduledBy')
```

```
        status = data.get('status') or 'AGENDADO'

        if not all([data_agendamento, hora_inicio, hora_fim, duracao_minutos, solenoide, usuario]):
            return jsonify({"error": "Campos obrigatorios faltando"}), 400

        sql = """
        INSERT INTO agendamentos (data_agendamento, hora_inicio, hora_fim, duracao_minutos, solenoide,
            usuario, status)
        VALUES (:data_agendamento, :hora_inicio, :hora_fim, :duracao_minutos, :solenoide, :usuario, :status)
        """
        result = db.session.execute(text(sql), {
            "data_agendamento": data_agendamento,
            "hora_inicio": hora_inicio,
            "hora_fim": hora_fim,
            "duracao_minutos": int(duracao_minutos),
            "solenoide": solenoide,
            "usuario": usuario,
            "status": status
        })
        db.session.commit()

        new_id = result.lastrowid
        if not new_id:
            new_id_row = db.session.execute(text("SELECT LAST_INSERT_ID()")).fetchone()
            new_id = new_id_row[0] if new_id_row else None

        return jsonify({
            "message": "Agendamento criado com sucesso",
            "id": new_id,
            "data_agendamento": data_agendamento,
            "hora_inicio": hora_inicio,
            "hora_fim": hora_fim,
            "duracao_minutos": int(duracao_minutos),
            "solenoide": solenoide,
            "usuario": usuario,
            "status": status
        }), 201
    except Exception as e:
        db.session.rollback()
        print(e)
        return jsonify({"error": f"Erro interno: {str(e)}"}), 500
```

```
@agendamento_routes.route('/agendamentos/<int:id>', methods=['PUT', 'POST'])
def update_agendamento(id):
    try:
        check_sql = "SELECT id FROM agendamentos WHERE id = :id"
        exists = db.session.execute(text(check_sql), {"id": id}).fetchone()
        if not exists:
            return jsonify({"error": "Agendamento nao encontrado"}), 404

        data = request.get_json(silent=True) or {}
        if not data:
            data = request.form.to_dict() or request.args.to_dict()

        data_agendamento = data.get('data_agendamento') or data.get('date')
        hora_inicio = data.get('hora_inicio') or data.get('startTime')
        hora_fim = data.get('hora_fim') or data.get('endTime')
        duracao_minutos = data.get('duracao_minutos') or data.get('durationMinutes')
        solenoide = data.get('solenoide') or data.get('solenoid')
        usuario = data.get('usuario') or data.get('scheduledBy')
        status = data.get('status')

        updates = []
        params = {"id": id}

        if data_agendamento is not None:
            updates.append("data_agendamento = :data_agendamento")
            params["data_agendamento"] = data_agendamento
        if hora_inicio is not None:
            updates.append("hora_inicio = :hora_inicio")
            params["hora_inicio"] = hora_inicio
        if hora_fim is not None:
            updates.append("hora_fim = :hora_fim")
            params["hora_fim"] = hora_fim
        if duracao_minutos is not None:
            updates.append("duracao_minutos = :duracao_minutos")
            params["duracao_minutos"] = int(duracao_minutos)
        if solenoide is not None:
            updates.append("solenoide = :solenoide")
            params["solenoide"] = solenoide
        if usuario is not None:
            updates.append("usuario = :usuario")
            params["usuario"] = usuario
        if status is not None:
```

```
            updates.append("status =
            :status") params["status"] =
            status

        if not updates:
            return jsonify({"message": "Nenhuma alteracao enviada"}), 200

        sql = f"UPDATE agendamentos SET {', '.join(updates)} WHERE id = :id"
        db.session.execute(text(sql), params)
        db.session.commit()
        return jsonify({"message": "Agendamento atualizado com sucesso"}), 200
    except Exception as e:
        db.session.rollback()
        print(e)
        return jsonify({"error": f"Erro interno: {str(e)}"}), 500

@agendamento_routes.route('/agendamentos/<int:id>', methods=['DELETE'])
def delete_agendamento(id):
    try:
        check_sql = "SELECT id FROM agendamentos WHERE id = :id" exists
        = db.session.execute(text(check_sql), {"id": id}).fetchone() if not
        exists:
            return jsonify({"error": "Agendamento nao encontrado"}), 404
```

7.1. A Função Auxiliar de Formatação de Dados *format_row()*

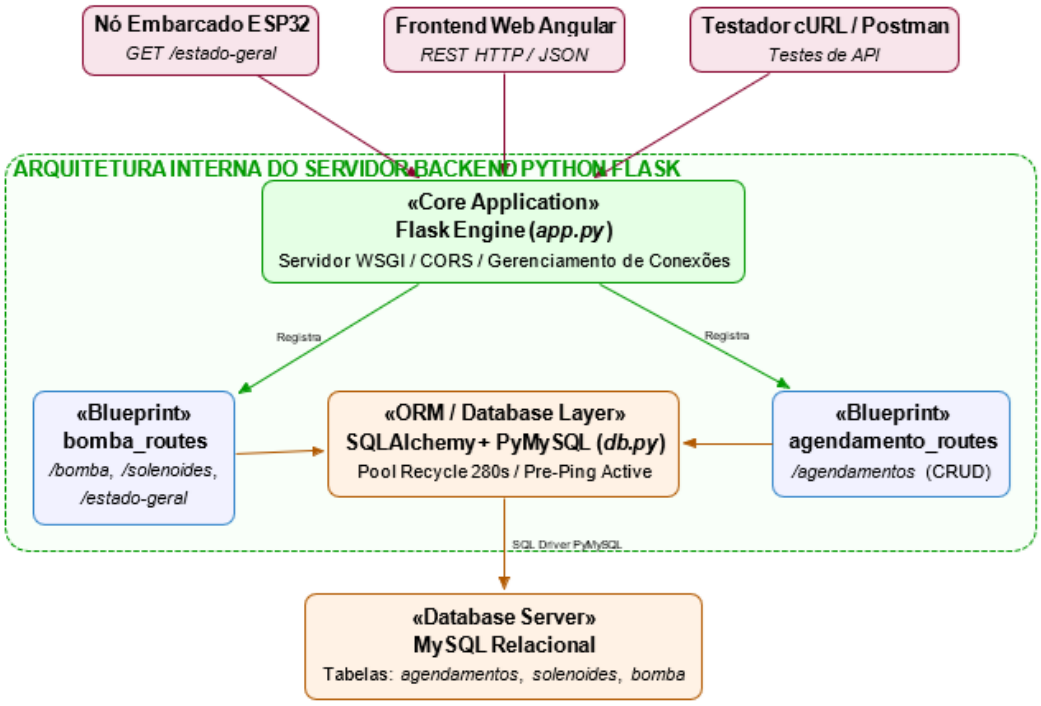
Ao consultar linhas no MySQL relacional, o driver PyMySQL devolve tuplas contendo objetos nativos de data (*datetime.date*) e tempo (*datetime.timedelta*). Como o formato de texto JSON não possui suporte nativo a objetos de data do Python, a função *format_row()* (linhas 8 a 34 da Listagem 3) realiza a conversão padronizada:

- Converte o objeto de data para a string padronizada ISO 8601 (*AAAA-MM-DD*);
- Calcula as horas, minutos e segundos a partir de objetos *timedelta*, devolvendo a string formatada *HH:MM:SS;*
- Garante que o documento JSON retornado pelo backend possua uma estrutura homogênea e amigável para a aplicação web em Angular.

8. MODELAGEM ARQUITETURAL DO BACKEND VIA DIAGRAMA DE COMPONENTES UML

Para formalizar a estrutura interna do servidor backend sob a ótica da engenharia de software, a Figura 1 apresenta o **Diagrama de Componentes UML** da aplicação Python Flask.

Figura 22: Diagrama de Componentes UML do servidor backend Flask em irrigação IoT.



Como ilustrado no diagrama da Figura 1, o servidor Flask atua como o ponto central de roteamento: ele recebe as requisições enviadas pelos clientes externos, encaminha o fluxo para o Blueprint especialista correspondente e executa as rotinas SQL seguras via SQLAlchemy na base de dados MySQL.

9. FERRAMENTAS DE TESTE DE APIS: O QUE SÃO cURL E POSTMAN?

Durante a fase de implantação ou manutenção de um sistema de irrigação automatizado, é fundamental verificar se o servidor backend está funcionando corretamente antes de conectar os painéis de campo ESP32 ou publicar a aplicação web. Se o sistema falhar, a equipe precisa saber se o problema está na fiação elétrica do campo, na interface web ou na API backend.

Para testar e validar o funcionamento das rotas HTTP da API de forma isolada, os engenheiros de software utilizam duas ferramentas indispensáveis: o **cURL** e o **Postman**.

9.1. O que é o cURL (*Client URL*)?

O cURL é uma ferramenta utilitária de linha de comando integrada nativamente a sistemas operacionais modernos (Linux, macOS e Windows Terminal).

Para o agrônomo, o cURL é o equivalente a um multímetro digital de rede: uma ferramenta simples e extremamente rápida que permite ao técnico enviar uma requisição HTTP crua diretamente do terminal de comandos para o servidor *www.meusitedeirrigacao.com* e ver na tela exatamente o texto JSON devolvido pelo Flask, juntamente com o código de status (ex: 200 OK ou 400 Bad Request).

9.2. O que é o Postman?

O **Postman** é um ambiente gráfico de desenvolvimento e testes de APIs.

Enquanto o cURL é utilizado via texto na linha de comando, o Postman oferece uma interface visual completa com botões, caixas de diálogo, abas para preenchimento de parâmetros, salvamento de coleções de testes e visualização formatada e colorida dos payloads JSON retornados pelo servidor. O Postman permite ao engenheiro simular visualmente a atuação de uma solenoide, cadastrar agendamentos de teste e validar respostas em segundos.

10. EXEMPLOS PRÁTICOS DE TESTES DE ROTAS COM cURL E POSTMAN

A seguir, apresentam-se exemplos práticos e reais de comandos cURL e requisições no Postman para testar cada uma das rotas do backend publicado em *www.meusitedeirrigacao.com*.

10.1. Teste 1: Consulta do Estado Geral (Consumo do ESP32)

- **Objetivo:** Verificar se a rota */estado-geral* está respondendo corretamente e atualizando a bomba e solenoides com base no horário atual.
- **Comando cURL no Terminal:**

```
curl -X GET "http://www.meusitedeirrigacao.com/estado-geral"
```

- **Configuração no Postman:**
  - Método: *GET*
  - URL: http://www.meusitedeirrigacao.com/estado-geral
  - Clique no botão **Send**.

- **Resposta Esperada do Servidor (HTTP 200 OK):**

```
{
  "bomba": true,
  "solenoides": [
    { "id": 1, "ativo": true },
    { "id": 2, "ativo": false },
    { "id": 3, "ativo": false },
    { "id": 4, "ativo": false }
  ]
}
```

10.2. Teste 2: Cadastro de Novo Agendamento de Irrigação

- **Objetivo:** Inserir uma nova rotina de rega para o setor do Tomate com duração de 20 minutos.
- **Comando cURL no Terminal:**

```
curl -X POST "http://www.meusitedeirrigacao.com/agendamentos" \
     -H "Content-Type: application/json" \
     -d '{
           "data_agendamento": "2026-09-30",
           "hora_inicio": "08:00:00",
           "hora_fim": "08:20:00",
           "duracao_minutos": 20,
           "solenoide": "Tomate",
           "usuario": "Thiago (Agronomo)"
         }'
```

- **Configuração no Postman:**
  - Método: *POST*
  - URL: http://www.meusitedeirrigacao.com/agendamentos
  - Aba **Headers**: Adicionar *Content-Type: application/json*
  - Aba **Body**: Selecionar a opção raw (formato JSON) e colar o objeto com os dados da rega.
  - Clique no botão **Send**.
- **Resposta Esperada do Servidor (HTTP 201 Created):**

```
{
  "id": 15,
  "message": "Agendamento criado com sucesso",
  "data_agendamento": "2026−09−30",
  "hora_inicio": "08:00:00",
  "hora_fim": "08:20:00",
  "duracao_minutos": 20,
  "solenoide": "Tomate",
  "usuario": "Thiago (Agronomo)",
  "status": "AGENDADO"
}
```

10.3. Teste 3: Acionamento Manual de Válvula Solenoide

- **Objetivo:** Forçar a abertura imediata da solenoide 1 (Setor Melão).
- **Comando cURL no Terminal:**

```
curl −X POST "http://www.meusitedeirrigacao.com/solenoide?id=1&ativo=true"
```

- **Configuração no Postman:**
  - Método: *POST*
  - URL: http://www.meusitedeirrigacao.com/solenoide?id=1&ativo=true
  - Clique no botão **Send**.

- **Resposta Esperada do Servidor (HTTP 200 OK):**

```
{
  "ativo": true,
  "message": "Solenoide 1 atualizado"
}
```

10.4. Teste 4: Atualização de Agendamento Existente

- **Objetivo:** Alterar a duração e o horário de término do agendamento ID 15.
- **Comando cURL no Terminal:**

```
curl −X PUT "http://www.meusitedeirrigacao.com/agendamentos/15" \
     −H "Content−Type: application/json" \
     −d '{
           "hora_fim": "08:30:00",
           "duracao_minutos": 30
         }'
```

- **Configuração no Postman:**
  - Método: *PUT*
  - URL: http://www.meusitedeirrigacao.com/agendamentos/15
  - Aba **Body**: Formato JSON contendo apenas os campos atualizados.
  - Clique no botão **Send**.
- **Resposta Esperada do Servidor (HTTP 200 OK):**

```
{
  "message": "Agendamento atualizado com sucesso"
}
```

10.5. Teste 5: Deleção de Agendamento Cancelado

- **Objetivo:** Excluir o agendamento ID 15 do banco de dados MySQL.
- **Comando cURL no Terminal:**

```
curl -X DELETE "http://www.meusitedeirrigacao.com/agendamentos/15"
```

- **Configuração no Postman:**
  - Método: *DELETE*
  - URL: http://www.meusitedeirrigacao.com/agendamentos/15
  - Clique no botão **Send**.
- **Resposta Esperada do Servidor (HTTP 200 OK):**

```
{
  "message": "Agendamento deletado com sucesso"
}
```

11. CONSIDERAÇÕES DE SEGURANÇA E DEPLOYMENT EM AMBIENTES DE PRODUÇÃO RURAL

Para garantir que o backend Python Flask opere com máxima estabilidade e segurança na infraestrutura de servidores da fazenda digital, devem-se observar as seguintes práticas de engenharia:

- **Servidor WSGI de Produção:** O servidor embutido do *Flask (app.run())* é voltado apenas para desenvolvimento e testes. Em produção no servidor corporativo *www.meusitedeirrigacao.com*, a aplicação deve ser executada através de um servidor WSGI profissional (como o Gunicorn ou uWSGI), gerenciado por um servidor web Nginx ou pelo utilitário *passenger_wsgi.py.*
- **Proteção CORS Restritiva:** Em ambiente de produção, a diretiva *CORS(app)* deve ser configurada para permitir requisições apenas vindo da origem autorizada do painel web da fazenda, impedindo que sites não autorizados na internet façam

requisições maliciosas para acionar solenoides no campo.

- **Variáveis de Ambiente para Credenciais:** As senhas do banco de dados MySQL e a chave secreta da aplicação *(app.secret_key*) nunca devem ser gravadas diretamente em código-fonte aberto, devendo ser injetadas dinamicamente através de variáveis de ambiente do sistema operacional *(os.environ).*

12. SÍNTESE DO CAPÍTULO E CONCLUSÃO GERAL DA OBRA

Este capítulo apresentou a arquitetura completa do servidor **Backend Python Flask**, que atua como o cérebro central de processamento e persistência do sistema hipotético de irrigação automatizada em Internet das Coisas.

Ficou demonstrada a importância da modularização de rotas via Blueprints, a gestão resiliente do banco de dados relacional MySQL através do SQLAlchemy com *pool recycling* e *pre-ping*, e o funcionamento do algoritmo inteligente da função *sincronizar_status()*, que avalia os agendamentos temporais e atualiza automaticamente os relés e a motobomba principal. Além disso, a arquitetura foi formalizada via Diagrama de Componentes UML e os testes práticos de rotas foram detalhados utilizando as ferramentas cURL e Postman no domínio corporativo *www.meusitedeirrigacao.com.*

Com o encerramento deste sétimo capítulo, esta obra consagra-se como um tratado abrangente, rigoroso e prático sobre a engenharia de sistemas de irrigação automatizados com IoT:

- **Capítulo 1:** Introdução às interfaces web e ecossistema de irrigação IoT;
- **Capítulo 2:** Especificações dos equipamentos, sensores capacitivos, válvulas solenoides e motobombas;
- **Capítulo 3:** Circuitos eletreletrônicos de amostragem, optoacopladores e chaves contatoras de potência;
- **Capítulo 4:** Fundamentos do protocolo HTTP REST e arquitetura cliente-servidor;
- **Capítulo 5:** Firmware C++ em microcontrolador ESP32, modelo de Cliente Magro (*Thin Client*) de Tanenbaum e salvaguardas falha-segura;
- **Capítulo 6:** Arquitetura Frontend SPA Angular, Programação Orientada a Objetos, Diagrama de Componentes UML, desacoplamento de backend e ecossistemas multicliente;
- **Capítulo 7:** Servidor Backend Python Flask, arquitetura de rotas RESTful, sincronização automática temporal, persistência MySQL relacional e testes de integração com cURL e Postman.

REFERÊNCIAS

MARTIN, Robert C. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. Boston: Prentice Hall, 2017.

SOMMERVILLE, Ian. **Software Engineering**. 10. ed. Boston: Pearson, 2016.

CAPÍTULO 8

CONCLUSÃO E PERSPECTIVAS FUTURAS

1. A JORNADA DA AUTOMAÇÃO NA AGRICULTURA

Chegamos ao fim de nossa jornada neste livro, mas este é, na verdade, apenas o começo de uma revolução muito maior que está transformando o campo e a forma como interagimos com o meio ambiente. Ao longo destas páginas, nós – Thiago, guiado pela visão e pelas necessidades agronômicas, e Sergio, trazendo a expertise e as ferramentas da tecnologia da informação – unimos nossas vivências e conhecimentos para demonstrar, de forma prática e tangível, como a Internet das Coisas (IoT) pode ser aplicada para resolver problemas reais da agricultura moderna.

A proposta de construir um sistema de irrigação automatizado, partindo do absoluto zero, não foi concebida apenas como um exercício acadêmico ou teórico. Pelo contrário, foi um convite aberto para que você, leitor, colocasse a mão na massa, experimentasse, errasse e aprendesse no processo. Vimos, passo a passo, como componentes eletrônicos simples e acessíveis, a exemplo do microcontrolador ESP32, relés e sensores capacitivos, podem ser transformados em ferramentas formidáveis de monitoramento e controle. Tudo isso se torna possível quando esses componentes são orquestrados por uma lógica de programação bem estruturada e apoiados por uma infraestrutura de comunicação em rede confiável.

Desde a compreensão dos conceitos mais básicos de eletrônica e da física por trás dos sensores, até a implementação complexa de um servidor backend e o desenvolvimento de uma interface gráfica amigável e responsiva (o frontend), percorremos todas as camadas necessárias para tirar um projeto de IoT do papel e levá-lo para a lavoura. Discutimos a importância crítica de medir a umidade do solo com precisão cirúrgica, a necessidade de controlar atuadores mecânicos – como válvulas solenoides e bombas d'água – de forma segura e autônoma, e, acima de tudo, a relevância de ter todos esses dados processados e disponíveis na palma da mão do produtor, em tempo real, independentemente de onde ele esteja.

Esta jornada evidenciou que a tecnologia pela tecnologia não é suficiente. Ela precisa de um propósito. E no contexto da irrigação, o propósito é claro: otimizar o uso da água, garantir o desenvolvimento saudável das culturas e facilitar a tomada de decisão. O sistema que construímos não é uma caixa preta inacessível, mas sim uma plataforma aberta, modular e expansível, projetada para crescer junto com as necessidades do usuário e as demandas da plantação.

2. A SINERGIA ENTRE AGRONOMIA E TECNOLOGIA DA INFORMAÇÃO

Um dos grandes diferenciais da agricultura contemporânea – frequentemente chamada de Agricultura 4.0 ou Agricultura Digital – é a necessidade inerente de colaboração entre áreas que, no passado, raramente dialogavam. O desenvolvimento deste livro e do

sistema de irrigação nele proposto é o reflexo direto dessa necessidade.

A visão do agrônomo (Thiago) é fundamental para definir *o que* precisa ser feito. É o agrônomo quem entende as demandas hídricas da planta em suas diferentes fases fenológicas, quem conhece os tipos de solo, a evapotranspiração e os impactos do déficit ou do excesso de água no rendimento da cultura. Sem esse conhecimento de domínio, qualquer sistema automatizado corre o risco de aplicar água na hora errada, na quantidade errada, desperdiçando recursos e prejudicando a produção.

Por outro lado, a visão do profissional de TI (Sergio) é essencial para determinar como fazer. É a ciência da computação que fornece as ferramentas para capturar dados analógicos do ambiente e transformá-los em informações digitais transmitidas globalmente. A arquitetura de software, a escolha dos protocolos de comunicação (como HTTP e MQTT), a segurança da informação, a modelagem do banco de dados e a criação de interfaces de usuário intuitivas são as pontes que conectam a necessidade agronômica à solução tecnológica viável.

Quando essas duas áreas convergem, o resultado é um salto exponencial em eficiência. O produtor rural deixa de tomar decisões baseadas puramente no instinto ou na tradição, e passa a ser guiado por dados concretos (data-driven). A tecnologia de TI atua como uma extensão dos sentidos do agrônomo, permitindo-lhe “sentir” a umidade do solo em dezoito hectares simultaneamente e intervir instantaneamente com um simples toque na tela do celular. Esta sinergia não apenas moderniza a fazenda, mas cria um novo paradigma de gestão agrícola, onde a precisão substitui a aproximação.

3. IMPACTOS ECONÔMICOS E AMBIENTAIS: A SUSTENTABILIDADE NA PRÁTICA

Não se pode falar de automação de irrigação sem abordar frontalmente a questão da sustentabilidade. A água é, indiscutivelmente, o recurso mais crítico para a produção de alimentos e, simultaneamente, um recurso natural cada vez mais escasso e disputado globalmente. A agricultura é responsável pelo consumo da maior parcela de água doce do planeta, e métodos de irrigação ineficientes contribuem significativamente para o desperdício desse bem inestimável.

Ao implementar um sistema IoT como o descrito neste livro, os impactos positivos operam em duas frentes indissociáveis: a ambiental e a econômica.

Do ponto de vista ambiental, o sistema garante que a água seja fornecida às plantas estritamente quando necessário e na quantidade exata. Ao monitorar a umidade do solo em tempo real, evitamos a irrigação excessiva, que não apenas desperdiça água, mas também causa a lixiviação de nutrientes do solo (levando-os para camadas mais profundas onde as raízes não alcançam) e pode gerar escoamento superficial que polui mananciais. A conservação hídrica deixa de ser um conceito abstrato e torna-se uma prática diária, automatizada e quantificável. O respeito ao meio ambiente é, portanto, codificado no coração do sistema.

Sob a ótica econômica, a eficiência hídrica traduz-se diretamente em redução de

custos operacionais. Menos tempo de bomba ligada significa uma drástica diminuição no consumo de energia elétrica ou de combustíveis fósseis, que frequentemente representam uma fatia expressiva dos custos de produção em propriedades irrigadas. Além disso, a aplicação precisa de água otimiza a absorção de fertilizantes e reduz a incidência de doenças fúngicas associadas ao excesso de umidade, resultando em plantas mais saudáveis e, consequentemente, em maior produtividade e qualidade da colheita. Em suma, o produtor passa a produzir mais, utilizando menos recursos. O retorno sobre o investimento (ROI) em tecnologias de automação agrícola, quando bem dimensionadas e aplicadas, costuma ser rápido e consistente, comprovando que sustentabilidade e lucratividade podem e devem caminhar lado a lado.

4. DESAFIOS E LIÇÕES APRENDIDAS NA IMPLEMENTAÇÃO

Embora a teoria seja elegante, a prática no campo é repleta de desafios, e seria irrealista concluir este guia sem abordá-los com honestidade. Desenvolver e implantar hardware e software para ambientes agrícolas (outdoor) exige considerações que vão muito além daquelas necessárias para um ambiente controlado de escritório ou laboratório.

A primeira grande lição diz respeito à resiliência do hardware. O ambiente agrícola é implacável. Os equipamentos estão expostos a variações extremas de temperatura, radiação solar intensa (UV), chuva, poeira, insetos e, em muitos casos, produtos químicos corrosivos. Um protótipo que funciona perfeitamente em uma protoboard na mesa de trabalho falhará em poucos dias no campo se não for adequadamente protegido. Investir em caixas herméticas (com grau de proteção IP adequado), prensa-cabos de qualidade, isolamento contra umidade e proteção contra surtos elétricos é não apenas recomendável, mas obrigatório para a longevidade do sistema.

A conectividade é outro gargalo crônico. Enquanto áreas urbanas desfrutam de internet de alta velocidade e redes Wi-Fi estáveis, o cenário rural frequentemente sofre com a ausência de sinal de telefonia celular e infraestrutura de rede precária. O uso de Wi-Fi, como abordado em parte de nosso projeto, é excelente para estufas ou áreas próximas à sede da fazenda. No entanto, para grandes extensões de terra, a conectividade dita as regras do jogo. É aqui que compreendemos a necessidade de planejar a infraestrutura de comunicação desde o dia zero, aceitando que a perda de conexão será uma ocorrência comum e que o sistema (edge computing) deve ser capaz de operar de forma autônoma e segura mesmo quando desconectado do servidor central.

Por fim, a manutenção contínua é uma realidade inexorável. Sensores descalibram, cabos se rompem por ação de animais ou máquinas, e servidores precisam de atualizações de segurança. Um sistema IoT não é um produto que se instala e se esquece; é um organismo vivo que requer monitoramento, ajustes e melhorias iterativas. A lição mais valiosa que aprendemos é que a simplicidade da manutenção deve ser uma premissa de design. Códigos bem documentados, facilidade para substituição física de componentes (plug-and-play) e logs de sistema claros são os melhores amigos do mantenedor a longo

prazo.

5. O FUTURO DA IRRIGAÇÃO E DA AGRICULTURA DIGITAL

O projeto que desenvolvemos ao longo dos capítulos fornece uma fundação sólida, mas as possibilidades de expansão e evolução são virtualmente infinitas. A tecnologia não para, e a agricultura digital está apenas em sua infância. Ao olharmos para o futuro da irrigação e da gestão agrícola, diversas tendências tecnológicas se apresentam como os próximos passos naturais para sistemas como o nosso.

5.1 Redes LPWAN (LoRaWAN e Sigfox)

Para contornar o problema da conectividade em grandes propriedades, as redes LPWAN (*Low Power Wide Area Network*) representam o padrão ouro atual. Tecnologias como LoRaWAN permitem que sensores de umidade do solo, operando com baterias pequenas e painéis solares minúsculos, transmitam dados por quilômetros de distância até um gateway central, consumindo uma fração minúscula de energia. A integração de módulos LoRa ao nosso nó sensor ESP32 permitiria cobrir áreas vastas da fazenda sem a necessidade de infraestrutura complexa de cabeamento ou repetidores Wi-Fi.

5.2 Inteligência Artificial e Machine Learning

Atualmente, nosso sistema opera com base em regras *hard-coded* (limiares fixos de umidade que ligam ou desligam a bomba). O futuro pertence aos sistemas preditivos e adaptativos. Com o acúmulo de dados históricos ao longo de meses ou anos – umidade, temperatura, precipitação, evapotranspiração, estágios da cultura – podemos treinar modelos de Machine Learning (Aprendizado de Máquina) (KAMILARIS; PRENAFETA-BOLDÚ, 2018; WOLFERT et al., 2017; FERENTINOS, 2018).

Um sistema dotado de Inteligência Artificial (IA) não apenas reagiria à falta de água no solo, mas anteciparia a necessidade de irrigação baseando-se na previsão do tempo (vai chover amanhã, logo não é necessário irrigar hoje) e no padrão histórico de consumo da planta. A IA seria capaz de identificar anomalias, como um vazamento na tubulação (pressão caindo, mas umidade não subindo) ou uma bomba defeituosa, alertando o produtor preventivamente antes que o dano ocorra.

5.3 Integração com Drones e Imagens de Satélite

A agricultura de precisão está se tornando cada vez mais visual. Sensores de solo fornecem dados pontuais e extremamente precisos, mas imagens multiespectrais capturadas por drones ou satélites fornecem uma visão macro da saúde da lavoura. A integração dos dados de umidade do nosso sistema IoT com mapas de índice de vegetação (como o NDVI) permitirá a criação de zonas de manejo hídrico variáveis. Em vez de irrigar um talhão inteiro de maneira uniforme, o sistema poderá controlar válvulas setoriais para aplicar mais água em manchas do terreno onde o solo retém menos umidade e a planta apresenta estresse,

e menos água em áreas que já estão adequadamente úmidas, otimizando ainda mais o uso dos recursos.

5.4 Edge Computing (Computação de Borda)

À medida que adicionamos mais sensores e exigimos reações mais rápidas, depender exclusivamente do processamento na nuvem pode introduzir latência indesejada e vulnerabilidade a falhas de rede. A tendência de *Edge Computing* (processamento na borda) envolverá colocar controladores mais potentes, como um Raspberry Pi, diretamente no campo. Esses dispositivos atuariam como cérebros locais, capazes de executar algoritmos complexos de IA e tomar decisões de irrigação críticas instantaneamente, sincronizando os dados com a nuvem apenas quando a conexão estiver disponível (SHI et al., 2016; SATYANARAYANAN, 2017).

5.5 Protocolo MQTT (*Message Queuing Telemetry Transport*)

Embora o protocolo HTTP (*polling*) adotado ao longo do livro seja amplamente consolidado e didático, arquiteturas IoT de alto volume e baixo consumo energético frequentemente migram para o protocolo MQTT (OASIS, 2019; AL-FUQAHA et al., 2015). Diferentemente das custosas requisições síncronas HTTP, o MQTT opera sob o paradigma publish/subscribe através de intermediários (*brokers*), resultando em menor sobrecarga de rede, suporte a níveis de qualidade de serviço (QoS) e notificação bidirecional assíncrona (HUNKELER; TRUONG; STANDFORD-CLARK, 2008). Essa transição é altamente recomendada quando a propriedade agrícola atinge a escala de centenas ou milhares de sensores autônomos operando simultaneamente.

6. PALAVRAS FINAIS

Chegamos ao fim deste guia com a esperança de termos desmistificado a criação de sistemas IoT para a agricultura. O que apresentamos aqui é mais do que um manual técnico; é uma filosofia de trabalho. Acreditamos firmemente que o conhecimento deve ser compartilhado e que a tecnologia deve ser uma força democratizadora, acessível tanto ao grande produtor rural quanto ao agricultor familiar, ao estudante universitário, ao hobbista e ao pesquisador.

O sistema de irrigação que construímos juntos – desde a soldagem dos primeiros fios, passando pela escrita das linhas de código C++ e Python, até o design da interface no navegador – é uma prova de que não é necessário depender exclusivamente de soluções comerciais fechadas e de alto custo para inovar no campo. A revolução do código aberto (*open source*) e do hardware livre (*open hardware*) empodera indivíduos para criarem soluções sob medida para seus próprios problemas.

Esperamos que o conteúdo abordado tenha servido como um catalisador para a sua criatividade. Encorajamos você a pegar este projeto base, desconstruí-lo, melhorá-lo e adaptá-lo à sua realidade específica. Adicione novos sensores, crie lógicas mais complexas,

projete um aplicativo mobile nativo, integre-o a assistentes virtuais de voz. O limite é ditado apenas pela sua imaginação e pela disposição em aprender e experimentar.

Em nome do Thiago, cuja paixão pela terra e pelo cultivo impulsiona a busca por eficiência agronômica, e do Sergio, cuja dedicação à tecnologia e ao desenvolvimento de sistemas transforma ideias em realidade funcional, agradecemos imensamente por nos acompanhar nesta leitura.

Que as sementes do conhecimento plantadas através deste livro possam germinar, crescer e render frutos em abundância, na forma de sistemas mais inteligentes, fazendas mais sustentáveis e um futuro tecnológico mais brilhante e integrado para a agricultura.

Mãos à obra e boas colheitas, digitais e reais!

REFERÊNCIAS

FERENTINOS, Konstantinos P. Deep learning models for plant disease detection and diagnosis. **Computers and Electronics in Agriculture**, v. 145, p. 311–318, 2018. DOI: 10.1016/j.compag.2018.01.009.

AL-FUQAHA, Ala et al. Internet of Things: A survey on enabling technologies, protocols, and applications. **IEEE Communications Surveys and Tutorials**, v. 17, n. 4, p. 2347–2376, 2015. DOI: 10.1109/COMST.2015.2444095.

HUNKELER, Urs; TRUONG, Hong Linh; STANDFORD-CLARK, Andy. MQTT-S: A publish/subscribe protocol for wireless sensor networks. **2008 3rd International Conference on Communication Systems Software and Middleware and Workshops**, p. 791–798, 2008. DOI: 10.1109/COMSWA.2008.4554519.

KAMILARIS, Andreas; PRENAFETA-BOLDÚ, Francesc X. Deep learning in agriculture: A survey. **Computers and Electronics in Agriculture**, v. 147, p. 70–90, 2018. DOI: 10.1016/j.compag.2018.02.016.

OASIS. **MQTT Version** 5.0. [S.l.], 2019. Disponível em: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>.

SATYANARAYANAN, Mahadev. The emergence of edge computing. **Computer**, v. 50, n. 1, p. 30–39, 2017. DOI: 10.1109/MC.2017.9.

SHI, Weisong et al. Edge computing: Vision and challenges. **IEEE Internet of Things Journal**, v. 3, n. 5, p. 637–646, 2016. DOI: 10.1109/JIOT.2016.2579198.

WOLFERT, Sjaak et al. Big data in smart farming: A review. **Agricultural Systems**, v. 153, p. 69–80, 2017. DOI: 10.1016/j.agsy.2017.01.023.

ÍNDICE REMISSIVO

A

agricultura 39, 40, 54, 92, 93, 95, 96, 97

Agricultura 4.0 92

agronomia 48, 58

agrônomos 40

água 40, 41, 49, 50, 55, 60, 92, 93, 94, 95

ambiente agrícola 57, 94

B

backend 38, 39, 48, 49, 57, 80, 86, 87, 90, 91, 92

Backend 91

banco de dados 39, 80, 90, 91, 93

biologia vegetal 56

bomba hidráulica 41

C

campo 39, 40, 41, 52, 54, 56, 60, 87, 91, 92, 94, 96

Ciência da Computação 55

circuitos eletrônicos 39, 55

códigos 38, 39

consumo energético 40, 96

D

DELETE 90

E

ecossistema 91

ecossistema de irrigação IoT 91

eletrônica 40, 54, 92

equipamentos 39, 91, 94

evapotranspiração 41, 93, 95

F

física 40, 53, 92, 94

frontend 39, 56, 58, 59, 60, 92

G

GET 41, 51, 80, 88

H

HTTP 38, 39, 40, 41, 50, 51, 54, 55, 61, 87, 88, 89, 90, 91, 93, 96

I

infraestrutura tecnológica 39
interface gráfica 61, 92
interfaces web 39, 91
irrigação 39, 40, 41, 48, 49, 53, 55, 56, 57, 58, 59, 86, 87, 91, 92, 93, 95, 96
irrigação automatizada 40, 91

L

lavoura 40, 48, 53, 55, 56, 59, 80, 92, 95

M

microcontroladores 40, 41
microcontrolador ESP32 40, 49, 55, 91, 92
monitoramento 92, 94
motobombas 56, 91

P

plantas cultivadas 56
POST 88, 89
produtor rural 56, 58, 93, 96
programação 48, 92
protocolo 39, 40, 55, 61, 91, 96
PUT 90

R

redes 54, 57, 80, 94, 95

S

sensores de umidade 95
servidor 38, 39, 40, 41, 42, 48, 49, 51, 54, 55, 57, 61, 80, 86, 87, 90, 91, 92, 94
sistema em chip 40
sistemas computacionais 56
software 40, 86, 87, 93, 94
solo 40, 41, 49, 56, 92, 93, 95

T

técnico agrícola 56
tecnologia 92, 93, 95, 96, 97

V

válvulas solenoides 40, 91, 92

W

Web 56, 57, 60, 61



contato@editoraomnisscientia.com.br
https://editoraomnisscientia.com.br/
@editora_omnis_scientia
https://www.facebook.com/omnis.scientia.9
+55 87 99920-5762

EDITORA
OMNIS SCIENTIA

contato@editoraomnisscientia.com.br
https://editoraomnisscientia.com.br/
@editora_omnis_scientia
https://www.facebook.com/omnis.scientia.9
+55 87 99920-5762