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.