BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

INSERÇÃO AUTOMATIZADA DE SENSORES POR MEIO DE SIMULAÇÃO EM UMA ARQUITETURA MODEL-VIEW-CONTROLLER (MVC)

IVAN ALVES DE JESUS JÚNIOR

Rio Verde, GO

2026

INSTITUTO FEDERAL GOIANO - CAMPUS RIO VERDE
BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

INSERÇÃO AUTOMATIZADA DE SENSORES POR MEIO DE SIMULAÇÃO EM UMA ARQUITETURA MODEL-VIEW-CONTROLLER (MVC)

IVAN ALVES DE JESUS JÚNIOR

Trabalho de Conclusão de Curso apresentado ao Instituto Federal Goiano - Campus Rio Verde, como requisito parcial para a obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Adriano Soares de Oliveira Bailão

Rio Verde, GO

11 de março de 2026

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema Integrado de Bibliotecas do IF Goiano - SIBi

Alves de Jesus Junior, Ivan
J95 Inserção automatizada de sensores por meio de simulação em uma Arquitetura model-View-Controller (MVC) / Ivan Alves de Jesus Junior. Rio verde 2026.

26f. il.

Orientador: Prof. Dr. Adriano Soares de Oliveira Bailão.
Tcc (Bacharel) - Instituto Federal Goiano, curso de 0219201 - Bacharelado em Ciência da Computação - Integral - Rio Verde (Campus Rio Verde).
I. Título.

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO

PARA DISPONIBILIZAR PRODUÇÕES TÉCNICO-CIENTÍFICAS NO REPOSITÓRIO INSTITUCIONAL DO IF GOIANO

Com base no disposto na Lei Federal nº 9.610, de 19 de fevereiro de 1998, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia Goiano a disponibilizar gratuitamente o documento em formato digital no Repositório Institucional do IF Goiano (RIIF Goiano), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IF Goiano.

IDENTIFICAÇÃO DA PRODUÇÃO TÉCNICO-CIENTÍFICA

- ☐ Tese (doutorado)
- ☐ Dissertação (mestrado)
- ☐ Monografia (especialização)
- ☑ TCC (graduação)
- ☐ Artigo científico
- ☐ Capítulo de livro
- ☐ Livro
- ☐ Trabalho apresentado em evento
- ☐ Produto técnico e educacional - Tipo:

Nome completo do autor: Ivan Alves de Jesus Junior

Matrícula: 2018102201940341

Título do trabalho: Inserção automatizada de sensores por meio de simulação em uma Arquitetura model-View-Controller (MVC)

RESTRIÇÕES DE ACESSO AO DOCUMENTO

Documento confidencial: ☑ Não ☐ Sim, justifique:

Informe a data que poderá ser disponibilizado no RIIF Goiano: 11 / 03 / 2026

O documento está sujeito a registro de patente? ☐ Sim ☑ Não

O documento pode vir a ser publicado como livro? ☐ Sim ☑ Não

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O(a) referido(a) autor(a) declara:

• Que o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;

• Que obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autoria, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia Goiano os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;

• Que cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia Goiano.

gov.br Documento assinado digitalmente
IVAN ALVES DE JESUS JUNIOR
Data: 28/04/2026 18:50:52-0300
Verifique em https://validar.iti.gov.br

Rio verde — Local

11 / 03 / 2026 — Data

_____
Assinatura do autor e/ou detentor dos direitos autorais

Ciente e de acordo:

gov.br Documento assinado digitalmente
ADRIANO SOARES DE OLIVEIRA BAILAO
Data: 28/04/2026 20:45:57-0300
Verifique em https://validar.iti.gov.br

_____
Assinatura do(a) orientador(a)

Regulamento de Trabalho de Conclusão de Curso (TCC) – IF Goiano - Campus Rio Verde

ANEXO V - ATA DE DEFESA DE TRABALHO DE CURSO

Aos 11 dias do mês de março de dois mil e vinte e seis, às 08h00, reuniu-se a Banca Examinadora composta por: Prof. Dr. Adriano Soares de Oliveira Bailão (orientador), Prof. Me. Caíke da Rocha Damke e Profa. Me. Andrea Barboza Proto Sardi, para examinar o Trabalho de Conclusão de Curso (TCC) intitulado “SIMUSENSOR - ARQUITETURA MODEL-VIEW-CONTROLLER (MVC) PARA INSERÇÃO AUTOMATIZADA DE SENSORES POR MEIO DE SIMULAÇÃO”, de IVAN ALVES DE JESUS JUNIOR, estudante do curso de Ciência da Computação do IF Goiano – Campus Rio Verde, sob Matrícula nº 2018102201940341.

A palavra foi concedida ao estudante para a apresentação oral do TC, em seguida houve arguição do candidato pelos membros da Banca Examinadora. Após tal etapa, a Banca Examinadora decidiu pela APROVAÇÃO do estudante.

Ao final da sessão pública de defesa foi lavrada a presente ata, que, após apresentação da versão corrigida do TC, foi assinada pelos membros da Banca Examinadora.

Rio Verde, 11 de março de 2026.

(Assinado eletronicamente)

Adriano Soares de Oliveira Bailão

Orientador(a)

(Assinado eletronicamente)

Caike da Rocha Damke

Membro da Banca Examinadora

(Assinado eletronicamente)

Andrea Barboza Proto Sardi

Membro da Banca Examinadora

Documento assinado eletronicamente por:

- **Adriano Soares de Oliveira Bailao**, **PROFESSOR ENS BASICO TECN TECNOLOGICO** , em 11/03/2026 09:04:45.
- **Caike da Rocha Damke** , **PROFESSOR ENS BASICO TECN TECNOLOGICO** , em 11/03/2026 09:07:53.
- **Andrea Barboza Proto Sardi**, **PROFESSOR ENS BASICO TECN TECNOLOGICO** , em 11/03/2026 09:14:54.

Este documento foi emitido pelo SUAP em 10/03/2026. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifgoiano.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 798620
Código de Autenticação: d155ff81e8

“Para que todos vejam e compreendam que a mão do Senhor guiou cada passo desta caminhada.” (Isaías 41:20)

E à minha mãe, cuja força, orações e amor me sustentaram quando eu fraquejei, esta conquista também é sua.

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela resiliência e força concedidas durante esta caminhada. À minha família, por todo o apoio incondicional.

Aos meus amigos Camila Gabriele, João Victor Santana, Fernando Gonzaga, Pedro Severo, Lucas Costa, Matheus Joseph, Mateus Farias e Athos José, obrigado por sempre me apoiarem e acreditarem em mim. Um agradecimento especial à pequena Alice Damasceno, que, com sua inocência, foi fonte de paz e alegria nos momentos mais difíceis.

Ao meu orientador, Prof. Dr. Adriano Bailão, agradeço imensamente pela oportunidade de ser seu orientando e pela confiança depositada no meu trabalho. Ao meu colega de laboratório, Prof. Me. Caíke Damke, sou grato pelos conselhos valiosos e por estar sempre solícito.

RESUMO

Júnior, Ivan. **Inserção automatizada de sensores por meio de simulação em uma Arquitetura model-View-Controller (MVC).** 11 de março de 2026. 26 f. TCC – (Curso de Bacharel em Ciência da Computação), Instituto Federal Goiano - Campus Rio Verde. Rio Verde, GO.

Devido aos crescentes avanços na Internet das Coisas (IoT), este trabalho apresenta o SimuSensor, um backend baseado na arquitetura Model-View-Controller (MVC), desenvolvido para simular a inserção automatizada de dados de sensores de temperatura em uma estrutura de estufa agrícola. A cada segundo, o sistema gera e transmite, via requisição HTTP, valores de temperatura aleatórios ao controlador, que os processa e os armazena em um arquivo XML. O backend oferece uma base modular e escalável para o desenvolvimento de soluções de monitoramento e registro de dados em aplicações de Internet das Coisas (IoT).

Palavras-chave: Arquitetura MVC; Simulador; Sensores; Automatização; back-end.

ABSTRACT

Júnior, Ivan. Automated insertion of sensors through simulation in a Model-View-Controller (MVC) architecture.. 11 de março de 2026. 26 f. Trabalho de Conclusão de Curso – Bacharel em Ciência da Computação, Instituto Federal Goiano - Campus Rio Verde. Rio Verde, GO, 11 de março de 2026.

Due to the growing advances in the Internet of Things (IoT), this work presents SimuSensor, a backend system based on the Model-View-Controller (MVC) architecture, developed to simulate the automated insertion of temperature sensor data in an agricultural greenhouse structure. Every second, the system generates and transmits, via HTTP request, random temperature values to the controller, which processes and stores them in an XML file. The backend provides a modular and scalable foundation for the development of monitoring and data logging solutions in Internet of Things (IoT) applications.

Keywords: MVC architecture; Simulator; Sensors; Automation; back-end.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

IoT	*Internet das Coisas*
MVC	*Arquitetura Model-View-Controller*
P&D	*Pesquisa e Desenvolvimento*
HTTP	*Hypertext Transfer Protocol*
XML	*Extensible Markup Language*
PHP	*Hypertext Preprocessor*
HTTPD	*Hypertext Transfer Protocol daemon*
NCSA	*National Center for Supercomputing Applications*
MQTT	*Message Queuing Telemetry Transport*
Nginx	*Servidor web, proxy reverso e balanceador de carga de alto desempenho*

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

A evolução tecnológica tem impulsionado o surgimento de novos ramos de pesquisa, proporcionando maior versatilidade na criação de soluções para problemas do cotidiano. Dentre essas inovações, destaca-se a Internet das Coisas (IoT), que, por meio da computação de baixo custo, possibilita a conectividade, o monitoramento e o gerenciamento de diversos dispositivos, como sensores, eletrodomésticos e sistemas embarcados.

Nesse cenário, a agricultura digital tem ganhado cada vez mais relevância no Brasil e no mundo, com o objetivo de otimizar a produção e facilitar a rotina do agricultor. Uma das aplicações importantes nesse contexto é o uso de estufas, amplamente empregadas na agricultura, em laboratórios e em ambientes industriais, onde o controle de variáveis como temperatura é essencial para garantir a qualidade e eficiência dos processos (ALVES et al., 2024).

Para o desenvolvimento e validação de soluções baseadas em IoT, é comum a utilização de simulações. A simulação consiste na representação computacional de sistemas reais, permitindo a análise e o teste de comportamentos sem a necessidade de implementação física imediata. Essa abordagem reduz custos, facilita experimentações e possibilita a validação inicial de sistemas em ambientes controlados.

Atualmente, ferramentas como o Wokwi (Wokwi, 2025) e o Tinkercad (Autodesk, 2025) são amplamente utilizadas para simulação de circuitos eletrônicos. No entanto, essas plataformas apresentam limitações quando aplicadas ao desenvolvimento de sistemas completos de monitoramento, especialmente no que se refere à integração com camadas de software, como front-end e bancos de dados, além da dificuldade em simular grandes volumes de dados provenientes de sensores em escala.

Além disso, a ausência de sensores físicos em ambientes de teste pode dificultar a validação de sistemas IoT, tornando necessária a utilização de soluções que simulem a geração contínua e automatizada de dados. Paralelamente, existe a demanda por arquiteturas de software organizadas, modulares e escaláveis, capazes de suportar o desenvolvimento e a evolução desses sistemas.

Diante desse contexto, este trabalho propõe o desenvolvimento do software SimuSensor, baseado na arquitetura Model-View-Controller (MVC), com o objetivo de simular a inserção automatizada de dados de sensores de temperatura. A proposta visa suprir as limitações das ferramentas existentes, oferecendo uma solução robusta e escalável para testes, validação e desenvolvimento de aplicações em Internet das Coisas. .

2 Fundamentação teórica

2.1 Internet das Coisas

A Internet das Coisas (IoT) pode ser definida como uma infraestrutura que estende a conectividade do ambiente digital ao mundo físico, permitindo a identificação, monitoramento e integração de objetos por meio da rede. Trata-se de um paradigma que viabiliza a comunicação entre dispositivos capazes de coletar, processar e transmitir dados, utilizando sensores, atuadores, softwares e tecnologias de comunicação (VERMESAN et al., 2011).

De acordo com (VILLAMIL; HERNáNDEZ; TARAZONA, 2020), a IoT atua como um facilitador tanto no desempenho industrial quanto na melhoria da qualidade de vida, permitindo que dispositivos operem de forma autônoma por meio de arquiteturas que integram dispositivos físicos, redes de comunicação e serviços em nuvem.

O termo IoT foi introduzido em 1999 por Kevin Ashton, no Auto-ID Center do MIT, inicialmente relacionado ao uso de tecnologias RFID para automação de processos logísticos. Entretanto, aplicações precursoras já haviam sido desenvolvidas anteriormente, como o experimento conduzido por John Romkey em 1990, no qual um dispositivo doméstico foi conectado à internet (CHAOUCHI, 2010).

Com o avanço tecnológico, a IoT evoluiu para um ecossistema complexo e altamente integrado. Segundo (REIS et al., 2020), a tecnologia passou a incorporar sistemas capazes de operar em tempo real, expandindo sua aplicação para ambientes industriais e agrícolas, onde há necessidade de integração entre componentes físicos e sistemas computacionais.

Embora dispositivos IoT apresentem custo unitário relativamente baixo, sua implementação em larga escala envolve desafios significativos, incluindo custos elevados de infraestrutura, manutenção e requisitos rigorosos de desempenho, como baixa latência e alta disponibilidade. Estudos indicam que essa expansão pode gerar um impacto econômico global entre \$5,5 trilhões e \$12,6 trilhões até 2030 (McKinsey Global Institute, 2021), impulsionada por um ecossistema com bilhões de dispositivos conectados (ZIKRIA et al., 2021).

Diante desse cenário, a validação de soluções IoT exclusivamente por meio de infraestrutura física torna-se complexa e financeiramente onerosa. Nesse contexto, a simulação computacional surge como uma abordagem essencial para testar, validar e otimizar sistemas antes de sua implementação real.

2.2 IoT na Agricultura Digital

A aplicação da IoT na agricultura digital fundamenta-se na integração de sensores e sistemas conectados para coleta e análise de dados em tempo real (CHATAUT; PHOUM-

MALAYVANE; AKL, 2023). Esse modelo permite a otimização de processos produtivos, contribuindo para o aumento da eficiência, da produtividade e da sustentabilidade no campo.

A agricultura digital representa uma evolução das práticas tradicionais, incorporando tecnologias como inteligência artificial, robótica e sistemas de informação. Segundo Fountas et al. (2020), essa abordagem possibilita o gerenciamento preciso das variabilidades do ambiente agrícola, promovendo tomadas de decisão mais assertivas.

Nesse contexto, sensores desempenham papel central na aquisição de dados. Entre os principais dispositivos utilizados, destacam-se sensores de temperatura e umidade do solo, sensores atmosféricos e sensores químicos, responsáveis por monitorar condições essenciais ao desenvolvimento das culturas.

Considerando a criticidade desses dados, torna-se necessário garantir sua confiabilidade antes da implementação em campo. No entanto, a validação baseada exclusivamente em sensores físicos apresenta limitações, especialmente em termos de custo, escalabilidade e disponibilidade de equipamentos.

Dessa forma, a simulação computacional configura-se como uma estratégia fundamental no desenvolvimento de sistemas IoT. Essa abordagem permite reproduzir o comportamento de sensores e ambientes reais em cenários controlados, possibilitando a realização de testes sem a necessidade de infraestrutura física (SOMMERVILLE, 2011).

Entre os principais benefícios da simulação, destacam-se a redução de custos operacionais, a identificação antecipada de falhas e a possibilidade de testar diferentes cenários de forma segura e controlada. Além disso, a simulação viabiliza a geração de dados sintéticos em larga escala, aspecto essencial para o desenvolvimento e validação de sistemas de monitoramento.

Nesse contexto, torna-se evidente a necessidade de arquiteturas de software que suportem a organização, escalabilidade e manutenção desses sistemas. Assim, a utilização de padrões arquiteturais, como o Model-View-Controller (MVC), contribui para a separação de responsabilidades e para a construção de soluções modulares, facilitando o desenvolvimento de aplicações voltadas à simulação de sensores.

2.3 Arquitetura Model-View-Controller

A arquitetura Model-View-Controller (MVC) é um padrão de projeto amplamente utilizado no desenvolvimento de sistemas de software, cuja principal característica é a separação de responsabilidades em três componentes distintos. Esse padrão tem como objetivo reduzir o acoplamento entre as partes do sistema, facilitando a manutenção, escalabilidade e reutilização de código (VOORHEES, 2020).

A adoção do padrão MVC é especialmente relevante em sistemas que demandam organização estrutural e manipulação de dados em diferentes camadas, como aplicações voltadas à simulação e monitoramento de sensores em ambientes IoT.

A arquitetura é composta pelos seguintes elementos:

- **Model**: Representa a camada responsável pela lógica de negócio e gerenciamento dos dados. No contexto de sistemas IoT, essa camada é responsável pelo tratamento, armazenamento e simulação dos dados provenientes de sensores.
- **View**: Corresponde à camada de apresentação, responsável por exibir as informações ao usuário. Em aplicações de monitoramento, essa camada permite a visualização dos dados simulados, geralmente por meio de interfaces gráficas ou dashboards.
- **Controller**: Atua como intermediário entre o Model e a View, sendo responsável por receber as entradas do usuário, processá-las e acionar as regras de negócio apropriadas. Além disso, controla o fluxo da aplicação, definindo qual informação será apresentada ao usuário.

A utilização do padrão MVC em sistemas de simulação de sensores permite a separação clara entre a geração de dados (Model), sua apresentação (View) e o controle das interações (Controller). Essa abordagem contribui para a construção de sistemas mais organizados, modulares e adaptáveis, características essenciais em aplicações voltadas à Internet das Coisas.

3 Trabalhos relacionados

A literatura atual sobre agricultura digital e Indústria 4.0 no campo aponta para um consenso: a tecnologia é essencial para a segurança alimentar futura, porém sua adoção ainda enfrenta barreiras significativas relacionadas a custo, complexidade e infraestrutura. Para situar a proposta deste trabalho, foram analisados estudos que abrangem desde revisões teóricas e bibliométricas até implementações práticas baseadas em hardware.

Nesse contexto de evolução do setor, diversos autores têm se dedicado a mapear o estado da arte. Sott et al. (2021) realizaram uma análise bibliométrica abrangente, identificando que a Agricultura 4.0 evoluiu para integrar tecnologias como Internet das Coisas (IoT), *big data* e computação em nuvem, formando uma cadeia de valor agrícola altamente conectada. Os autores destacam que, enquanto a agricultura de precisão se concentra na variabilidade do campo, a Agricultura 4.0 amplia esse conceito ao incorporar sistemas inteligentes para gestão contextualizada.

Corroborando essa perspectiva, Abiri et al. (2023) afirmam que a agricultura digital utiliza tecnologias avançadas para otimizar sistemas produtivos e reduzir desperdícios, enfatizando o papel da automação e da robótica na substituição de processos manuais por operações contínuas e mais eficientes.

No que se refere às tecnologias habilitadoras dessa transformação, Issa et al. (2024) destacam a integração entre IoT, redes de comunicação e computação em nuvem como elementos fundamentais para atender à crescente demanda global por alimentos. Os autores ressaltam que o uso de redes de alta velocidade, como o 5G, viabiliza a transmissão de dados em tempo real e a tomada de decisão mais eficiente. Entretanto, persistem desafios relevantes: tanto Abiri et al. (2023) quanto Issa et al. (2024) apontam que os elevados custos de implementação inicial e as limitações de infraestrutura ainda representam obstáculos significativos, especialmente em países em desenvolvimento.

Além das questões estruturais, os sensores elementos fundamentais na coleta de dados também apresentam desafios técnicos. Queiroz et al. (2020) apresentam uma revisão detalhada sobre sensores aplicados ao solo, plantas e produtividade, evidenciando que o desempenho desses dispositivos é influenciado por múltiplos fatores físicos e químicos. Esse comportamento exige calibração frequente e aumenta a complexidade na obtenção de dados confiáveis diretamente em campo.

No âmbito das implementações práticas, Aditya et al. (2024) desenvolveram um sistema de monitoramento baseado em IoT, utilizando sensores de solo e estações meteorológicas. O sistema é capaz de coletar múltiplos parâmetros em tempo real, como temperatura, umidade, nutrientes (NPK) e pH, transmitindo os dados via protocolo MQTT para um *dashboard*. Embora o estudo demonstre viabilidade técnica, evidencia também a dependência de infraestrutura física específica e a necessidade de manutenção local,

reforçando as barreiras de custo e complexidade apontadas na literatura.

A análise conjunta desses trabalhos evidencia uma lacuna relevante. Enquanto estudos como os de Sott et al. (2021) e Abiri et al. (2023) concentram-se em aspectos teóricos e estratégicos, e Queiroz et al. (2020) abordam os aspectos técnicos dos sensores, trabalhos aplicados como o de Aditya et al. (2024) focam diretamente na implementação física. Nesse cenário, observa-se a ausência de soluções que permitam a validação de sistemas de monitoramento em nível de software, sem a necessidade de investimento imediato em hardware.

Diferentemente das abordagens que dependem da implementação física desde as etapas iniciais, este trabalho propõe uma arquitetura baseada em simulação. Essa abordagem permite testar cenários diversos e validar o comportamento do sistema antes da aquisição de dispositivos reais. Dessa forma, contribui para tornar o desenvolvimento de soluções em IoT mais viável economicamente, ampliando o acesso a tecnologias da Agricultura 4.0 por parte de estudantes, pesquisadores e pequenos produtores.

4 MATERIAIS E MÉTODOS

O fluxo de interação entre as camadas segue o padrão Usuário → View → Controller → Model → Controller → View, conforme ilustrado na Figura 1.

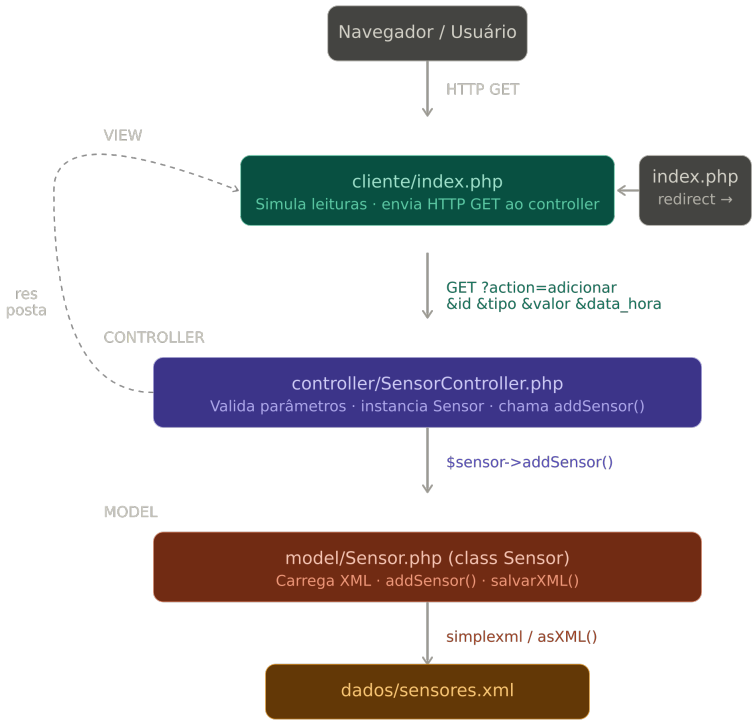

Figura 1 – Ilustração de uma arquitetura Model-View-Controller.

Neste projeto, foi desenvolvido um backend para a simulação de leituras de temperatura de um sensor instalado em uma casa de vegetação, utilizando a arquitetura Model-View-Controller (MVC).

Nesse backend, o arquivo `cliente/index.php` simula o envio contínuo de dados de temperatura ao controlador por meio de requisições HTTP GET, com valores gerados aleatoriamente a cada segundo. O controlador valida os parâmetros recebidos e aciona o *Model*, que persiste as leituras em um arquivo XML.

A estrutura de pastas e arquivos do projeto pode ser visualizada na Figura 2.

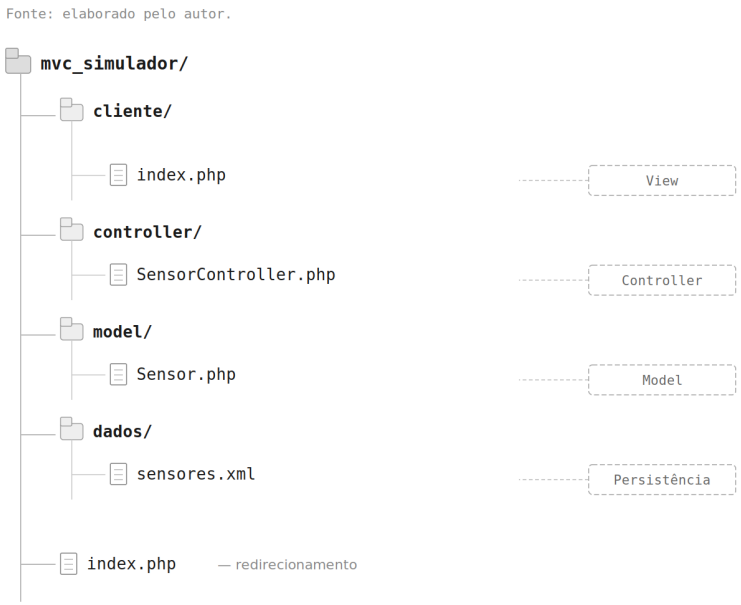

Figura 2 – Estrutura de pastas e arquivos do backend.

- **cliente/index.php**: Simula o papel da camada *View* da arquitetura MVC, gerando leituras de sensores e enviando-as ao controlador por meio de requisições HTTP GET.
- **controller/SensorController.php**: Implementa a camada *Controller*, sendo responsável por receber e validar os parâmetros da requisição e acionar o *Model* para persistir os dados.
- **model/Sensor.php**: Contém a classe que representa a camada *Model*, centralizando a lógica de acesso, leitura e escrita dos dados dos sensores no arquivo XML.
- **dados/sensores.xml**: Armazena os dados dos sensores no formato XML, garantindo a persistência das informações.
- **index.php**: Realiza o redirecionamento automático para a interface localizada na pasta `/cliente`.

Nas subseções 4.2 a 4.5 são apresentados o código-fonte e a explicação detalhada de cada arquivo que compõe a estrutura do sistema.

4.1 Tecnologias

4.1.1 PHP

O PHP (versão 8.2) é uma linguagem de programação *server-side* (lado do servidor), interpretada e executada diretamente no servidor web. Essa característica permite que o processamento de dados, a persistência de informações e a lógica de negócio sejam executados em um ambiente controlado, sem exposição ao cliente. Dessa forma,

o PHP viabiliza operações que linguagens *client-side*, como o JavaScript, não realizam adequadamente em razão das restrições de segurança impostas pelo ambiente de execução nos navegadores (BROOKS, 2009).

Historicamente, o PHP foi criado em 1994 por Rasmus Lerdorf. Inicialmente, a tecnologia não era uma linguagem completa, mas sim um conjunto de scripts *Common Gateway Interface* (CGI) escritos em C, utilizados para rastrear visitas ao currículo online do autor. Com a evolução da ferramenta, o PHP tornou-se amplamente adotado no desenvolvimento de aplicações web dinâmicas e robustas.

Principais características da linguagem PHP:

- **Execução no servidor (*server-side scripting*)**: O processamento ocorre no servidor web, e não no navegador do usuário. Isso permite a geração de páginas dinâmicas e o tratamento de formulários com conteúdo personalizado.
- **Integração com HTML**: O PHP pode ser incorporado diretamente em código HTML, o que o torna adequado para o desenvolvimento de interfaces web que demandam funcionalidades do lado do servidor.
- **Integração com bancos de dados**: A linguagem oferece suporte nativo à conexão com sistemas de banco de dados relacionais e não relacionais, sendo amplamente utilizada em aplicações que requerem armazenamento e recuperação de dados.
- **Compatibilidade entre plataformas**: O PHP é executado nos principais sistemas operacionais, incluindo Windows, Linux e macOS, e é compatível com servidores web como Apache e Nginx.

4.1.2 HTTP

A sigla HTTP corresponde a *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto). Esse protocolo é o principal responsável pela transferência de dados na Internet, estabelecendo as regras que regem o processo de requisição e resposta entre cliente e servidor. A comunicação é iniciada pelo cliente, normalmente um navegador ou outro agente de software que envia uma solicitação ao servidor, o qual retorna o recurso solicitado. O HTTP atua na camada de aplicação e define métodos específicos para a realização de requisições (GORALSKI, 2017).

Principais métodos de requisição HTTP:

- **GET**: Solicita um recurso identificado por uma URL. Os parâmetros são transmitidos na própria URL, o que torna esse método adequado para consultas e para a transmissão de dados não sensíveis em ambientes controlados.
- **HEAD**: Solicita apenas os cabeçalhos da resposta, sem o corpo do conteúdo. Utilizado para verificar metadados de um recurso sem transferi-lo integralmente.
- **POST**: Envia um bloco de dados ao servidor no corpo da requisição, sendo tipicamente utilizado para o envio de formulários ou para a criação de novos recursos em APIs.

4.1.3 HTTPD

O termo HTTPD é um acrônimo para *Hypertext Transfer Protocol Daemon*. Em sistemas operacionais multitarefa, um *daemon* refere-se a um processo executado em segundo plano, sem intervenção direta do usuário, aguardando e processando requisições de forma contínua. No contexto da infraestrutura web, conforme detalham Gourley e Totty (2002), o HTTPD é o componente de software responsável por escutar as portas de rede, gerenciar conexões, processar as solicitações HTTP recebidas e entregar recursos estáticos ou dinâmicos aos clientes.

4.1.4 Apache HTTP Server

O Apache HTTP Server, frequentemente denominado Apache, é um servidor web de código aberto criado em 1995 por um grupo de desenvolvedores com base no servidor NCSA HTTPD, desenvolvido por Rob McCool. Atualmente, é mantido pela Apache Software Foundation, organização responsável por uma ampla gama de projetos de software de código aberto voltados para infraestrutura e processamento distribuído na Internet.

O Apache pode ser definido como um servidor HTTPD robusto, compatível com as versões 1.1 e 2.0 do protocolo HTTP. Sua arquitetura é organizada de forma modular, permitindo que sua funcionalidade básica seja estendida por meio de módulos dinâmicos. Essa flexibilidade possibilita que desenvolvedores personalizem o comportamento do servidor de acordo com as necessidades específicas de cada aplicação.

4.1.5 XML

O XML (*Extensible Markup Language*) é um padrão de marcação que permite a estruturação e a representação de dados de forma hierárquica e autodescritiva. Sua principal característica reside na capacidade de possibilitar o intercâmbio e a interpretação de informações por sistemas heterogêneos como aplicações, bancos de dados e dispositivos embarcados (BRESSAN et al., 2003). O formato é regulamentado pelo *World Wide Web Consortium* (W3C) e incorpora características de outras linguagens, como o SGML e o HTML.

Ao longo dos anos, o XML consolidou-se como padrão para a integração entre sistemas, em razão de sua flexibilidade e da possibilidade de definir esquemas de validação específicos. Neste projeto, o XML é utilizado como mecanismo de persistência dos dados gerados pelo simulador, armazenando cada leitura de sensor em um arquivo estruturado.

- **Elementos e atributos XML**: Os elementos são as marcações que identificam e delimitam os dados, enquanto os atributos fornecem informações complementares sobre esses elementos. Juntos, definem a estrutura e o conteúdo do documento.

- **Esquemas XML (*XML Schemas*)**: Definem as regras estruturais que os documentos XML devem seguir, garantindo que os dados trocados entre sistemas obedeçam a um padrão consistente e validável.

4.2 Interface do Backend – cliente/index.php

Este arquivo é responsável por simular, na camada *View* da arquitetura MVC, o envio contínuo de leituras de temperatura ao controlador. A cada segundo, um valor de temperatura gerado aleatoriamente entre 20 e 35 é transmitido ao `SensorController.php` por meio de uma requisição HTTP GET, e a resposta recebida é exibida no navegador.

Vale observar que, neste projeto, a camada *View* acumula também a responsabilidade pela geração dos dados papel que, em uma implementação mais aderente ao MVC clássico, poderia ser delegado a um componente de serviço ou a um gerador de dados independente, mantendo a *View* restrita à apresentação. Essa decisão de projeto foi adotada por simplificação, dado o escopo de prótotipo do sistema; em uma evolução futura, essa responsabilidade poderia ser separada.

Em relação à escolha do método HTTP GET para o envio das leituras: em um contexto de produção, o envio de dados destinados à inserção seria realizado preferencialmente com o método POST, que transmite os parâmetros no corpo da requisição. Neste protótipo, o GET foi adotado pela simplicidade de implementação, já que a função `file_get_contents()` do PHP realiza requisições GET de forma direta e pelo fato de o sistema operar em ambiente fechado e controlado, sem exposição a dados sensíveis ou a usuários externos. Essa escolha é documentada aqui para diferenciação explícita entre o contexto de protótipo e uma implementação de produção mais robusta.

Algoritmo 4.1 – cliente/index.php

```
<?php

// Configuração de exibição de erros (recomendado apenas em
   ↪ desenvolvimento)
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

/*
 * Envia uma leitura de sensor ao controlador via requisição HTTP GET.
 * O GET foi escolhido pela simplicidade de implementação
 * com file_get_contents() em ambiente controlado.
 Em produção, recomenda-se o uso de POST para envio de dados destinados a
    ↪  inserção. Nota sobre urlencode(): a função garante que os valores
```

```
    ↪  seja corretamente transmitidos na URL (tratando espaços, acentos
    ↪ e caracteres especiais). Ela não tem finalidade de segurança
    ↪ contra ataques  neste protótipo, os dados são gerados internamente
    ↪  e não provm de entrada do usuário.
 */
function enviarLeitura($id, $tipo, $valor) {
    // Captura a data e hora atuais e codifica para uso seguro em URL
    $data_hora = urlencode(date('Y-m-d␣H:i:s'));
    $id = urlencode($id);
    $tipo = urlencode($tipo);
    $valor = urlencode($valor);

    // Monta a URL do controlador com os parâmetros da leitura
    $url = "http://localhost/mvc_simulador/controller/SensorController.php
       ↪ "
        . "?action=adicionar&id=$id&tipo=$tipo&valor=$valor&data_hora=
           ↪ $data_hora";

    // Envia a requisição GET e captura a resposta do controlador
    $resposta = file_get_contents($url);

    // Exibe no navegador o registro da leitura enviada e a resposta
       ↪ recebida
    echo "[$data_hora]␣Enviado:␣Sensor␣$id␣|␣$tipo␣=␣$valor␣=>␣Resposta:␣
       ↪ $resposta<BR>";

    // Força o envio imediato do buffer de saída para o navegador
    flush();
    ob_flush();
}

/**
 * Laço de simulação contínua.
 *
 * A cada iteração, gera um valor aleatório de temperatura (entre 20,0 e
    ↪ 35,0 graus)
 * e o envia ao controlador. A execução  pausada por 1 segundo entre cada
    ↪  envio.
 *
```

```
 * Atenção: cada aba ou janela do navegador que acessar esta página
    ↪ iniciará
 * um processo PHP independente, gerando fluxos de envio simultâneos.
 * Em um ambiente de produção, essa lógica deveria ser executada por um
    ↪ processo
 * dedicado no servidor (ex.: cronjob ou serviço em segundo plano),
    ↪ desvinculando
 * a geração de dados da camada de apresentação.
 */
while (true) {
    // Gera temperatura aleatória entre 20,0 e 35,0 graus Celsius
    enviarLeitura(1, "temperatura", rand(200, 350) / 10);
    sleep(1);
}
```

4.3 Controlador – controller/SensorController.php

O arquivo `SensorController.php` implementa a camada *Controller* da arquitetura MVC. Sua responsabilidade é receber as requisições HTTP provenientes da camada *View*, validar os parâmetros recebidos e acionar os métodos do *Model* para que os dados sejam persistidos. O controlador atua, portanto, como intermediário entre a interface de geração de dados e a lógica de acesso ao arquivo XML.

Como este componente expõe um *endpoint* acessível via HTTP, os cabeçalhos CORS (*Cross-Origin Resource Sharing*) são configurados para permitir requisições de qualquer origem decisão adequada ao contexto de protótipo em ambiente local. Em uma implementação voltada à produção, esses cabeçalhos deveriam ser restritos às origens autorizadas, e os parâmetros recebidos deveriam passar por validações mais rigorosas (como verificação de tipos, limites de valores e autenticação do cliente emissor). Neste protótipo, assume-se que os dados provêm de uma fonte confiável e controlada o próprio `cliente/index.php` o que justifica a ausência de validações mais detalhadas.

Algoritmo 4.2 – controller/SensorController.php

```
<?php
// Importa a classe Sensor (Model), garantindo que o arquivo seja
   ↪ incluido
// apenas uma vez durante a execucao
require_once __DIR__ . '/../model/Sensor.php';

// Cabecalhos CORS: permitem que qualquer origem acesse este endpoint via
   ↪  HTTP.
```

```php
// Em producao, substituir '*' pela origem especifica autorizada.
header("Access-Control-Allow-Origin: *");
header("Access-Control-Allow-Methods: POST, GET, PUT, DELETE, OPTIONS");
header("Access-Control-Allow-Headers: Content-Type");

// Instancia o Model responsavel pela persistencia dos dados
$sensor = new Sensor();

// Identifica o metodo HTTP utilizado na requisicao atual
$method = $_SERVER['REQUEST_METHOD'];

// Trata requisicoes OPTIONS (preflight do CORS), enviadas
    ↪ automaticamente
// pelo navegador antes de requisicoes com metodos como PUT, DELETE ou
    ↪ POST
// com corpo JSON para dominios diferentes
if ($method === 'OPTIONS') {
    http_response_code(200);
    exit;
}

// Endpoint de insercao de leitura via GET.
// Nota: o mtodo GET foi adotado neste prototipo pela simplicidade de
// integracao com file_get_contents() na View. Em producao, recomenda-se
// o uso de POST para operacoes de insercao de dados, pois o corpo da
// requisicao nao fica exposto na URL nem em logs de acesso do servidor.
//
// Os parametros recebidos sao tratados como confiaveis, pois o sistema
// opera em ambiente controlado. Em uma implementacao mais robusta,
// seria necessario validar tipos, intervalos de valores e autenticar
// a origem da requisicao.
if ($method === 'GET'
    && isset($_GET['action'])
    && $_GET['action'] === 'adicionar') {

    // Verifica se todos os parametros obrigatorios estao presentes
    if (isset($_GET['id'], $_GET['tipo'], $_GET['valor'], $_GET['data_hora']))
        ↪ {
```

```
        $id = $_GET['id'];
        $tipo = $_GET['tipo'];
        $valor = $_GET['valor'];
        // Decodifica a data/hora, revertendo a codificacao feita pela
            ↪ View
        $data_hora = urldecode($_GET['data_hora']);

        // Aciona o Model para persistir a leitura no arquivo XML
        $sensor->addSensor($id, $tipo, $valor, $data_hora);

        // Retorna confirmacao em texto simples
        header("Content-Type: text/plain");
        echo "$id $tipo $valor $data_hora";
        http_response_code(200);

    } else {
        // Parametros incompletos: retorna erro 400 (Bad Request)
        http_response_code(400);
        echo "Dados incompletos ou invalidos.";
    }

    exit;
}
?>
```

4.4 Model – model/Sensor.php

O arquivo `Sensor.php` implementa a camada *Model* da arquitetura MVC. Sua responsabilidade é encapsular toda a lógica de acesso, leitura e escrita dos dados dos sensores, que neste projeto são armazenados em um arquivo XML. O *Model* é acionado exclusivamente pelo *Controller*, mantendo o isolamento das camadas previsto pela arquitetura.

Algoritmo 4.3 – model/Sensor.php

```
<?php

class Sensor {

    // Caminho para o arquivo XML de persistencia
    private $xmlFile;
```

```
// Objeto SimpleXMLElement que representa o conteudo do arquivo XML em
    ↪ memoria
private $xml;

/**
 * Construtor: define o caminho do arquivo XML e carrega seu conteudo.
 * Caso o arquivo ainda nao exista, cria a estrutura inicial.
 */
public function __construct() {
    $this->xmlFile = __DIR__ . '/../dados/sensores.xml';

    if (file_exists($this->xmlFile)) {
        $this->xml = simplexml_load_file($this->xmlFile);
    } else {
        $this->criarEstruturaXML();
    }
}

/**
 * Cria a estrutura raiz do arquivo XML (<sensores></sensores>)
 * e salva o arquivo em disco.
 * Metodo privado: chamado apenas internamente pelo construtor.
 */
private function criarEstruturaXML() {
    $this->xml = new SimpleXMLElement('<sensores></sensores>');
    $this->salvarXML();
}

/**
 * Persiste o objeto XML atual no arquivo em disco.
 * Metodo privado: chamado apos cada alteracao no conteudo XML.
 */
private function salvarXML() {
    $this->xml->asXML($this->xmlFile);
}

/**
 * Adiciona uma nova leitura de sensor ao arquivo XML.
```

```
     *
     * @param mixed $sensor_id Identificador do sensor
     * @param string $tipo Tipo da leitura (ex.: "temperatura")
     * @param mixed $valor Valor numerico da leitura
     * @param string $data_hora Data e hora da leitura (formato Y-m-d H:i:
        ↪ s)
     */
    public function addSensor($sensor_id, $tipo, $valor, $data_hora) {
        // Garante que o XML esteja carregado antes de qualquer operacao
        if (!$this->xml) {
            $this->xml = simplexml_load_file($this->xmlFile);
        }

        // Adiciona o elemento <sensor> como filho do elemento raiz <
            ↪ sensores>
        $sensor = $this->xml->addChild('sensor');
        $sensor->addChild('id', $sensor_id);
        $sensor->addChild('tipo', $tipo);
        $sensor->addChild('valor', $valor);
        $sensor->addChild('data_hora', $data_hora);

        // Persiste as alteracoes no arquivo XML
        $this->salvarXML();
    }
}
?>
```

4.5 Redirecionador – index.php

O arquivo `index.php` localizado na raiz do projeto tem função exclusiva de redirecionamento: ao ser acessado, encaminha automaticamente o navegador para a interface do backend descrita na Seção 4.2 (`cliente/index.php`). Trata-se de um ponto de entrada único para o sistema, garantindo que qualquer acesso à raiz do projeto seja direcionado à camada *View* correta.

Algoritmo 4.4 – index.php (raiz)

```
<?php
// Redireciona automaticamente para a interface do backend em /cliente
header("Location: cliente/index.php");
exit;
```

```
?>
```

Cabe destacar que a arquitetura adotada neste protótipo propicia uma evolução natural do sistema em trabalhos futuros: a substituição do gerador de dados simulados por leituras provenientes de sensores reais (conectados via serial, MQTT ou outro protocolo) exigiria alterações apenas na camada *View* ou, de forma ainda mais aderente ao MVC, em um componente de serviço dedicado à aquisição de dados, sem necessidade de modificar o *Controller* ou o *Model*. Essa separação de responsabilidades é uma das principais vantagens práticas da arquitetura MVC para a extensibilidade do sistema.

5 Resultados

5.1 Resultado Científico

O presente projeto adota a arquitetura Model-View-Controller (MVC), amplamente utilizada no desenvolvimento de aplicações web por promover a separação de responsabilidades entre os componentes do sistema. As três camadas estão definidas da seguinte forma:

- **Model**: representada pelo arquivo `Sensor.php` (Seção 4.4), responsável pela manipulação direta dos dados dos sensores armazenados em XML.
- **View**: representada pelo arquivo `cliente/index.php` (Seção 4.2), responsável por simular o envio contínuo de leituras ao controlador e por exibir no navegador o registro das transmissões realizadas.
- **Controller**: implementada pelo arquivo `SensorController.php` (Seção 4.3), que atua como intermediário entre a camada *View* e a lógica de acesso aos dados.

A execução do sistema requer, inicialmente, a ativação do servidor Apache por meio do XAMPP (versão 8.2.12, que inclui o PHP 8.2 e o Apache 2.4), conforme demonstrado na Figura 3. Em seguida, deve-se copiar a pasta do projeto para o diretório `C:\xampp\htdocs`.

Vale observar que o painel do XAMPP exibe também os serviços MySQL e FTP, que aparecem listados por padrão na interface do aplicativo mas permanecem inativos neste projeto, o backend utiliza exclusivamente persistência em arquivo XML, sem qualquer dependência de banco de dados relacional ou protocolo de transferência de arquivos.

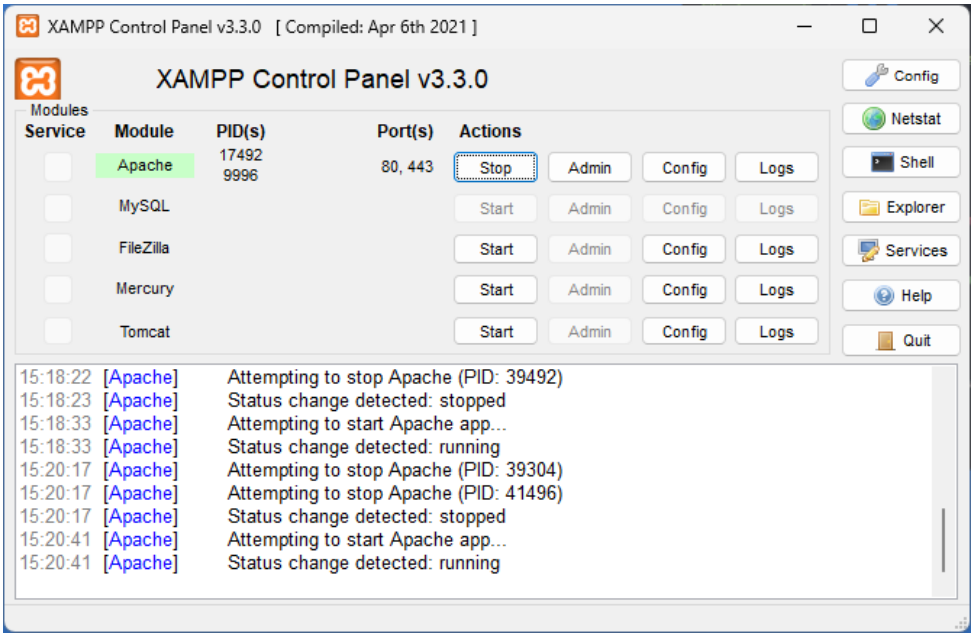

Figura 3 – Painel de controle do XAMPP com o Apache ativo (PHP 8.2 / Apache 2.4).

A Figura 4 demonstra o sistema em execução: após ativar o servidor Apache e acessar o endereço `http://localhost/mvc_simulador` no navegador, a camada *View* exibe continuamente as leituras de temperatura geradas aleatoriamente e processadas pelo `Sensor.php`.

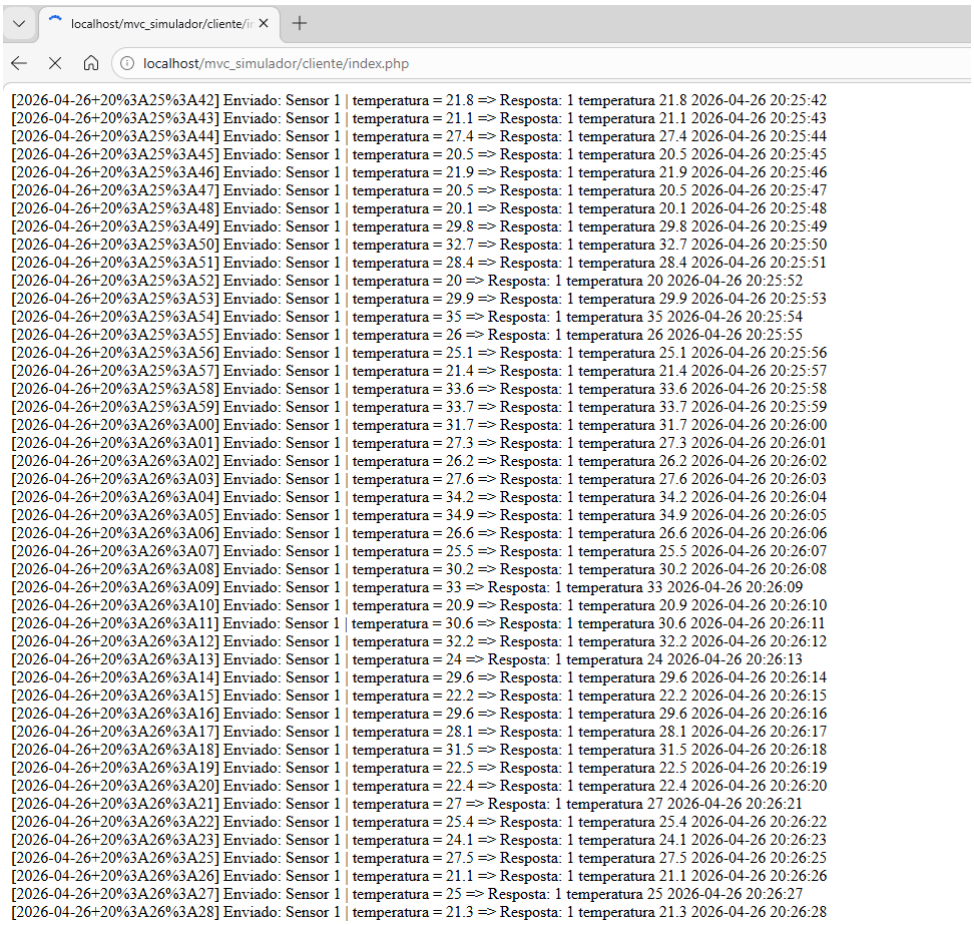

Figura 4 – Interface do backend exibindo as leituras geradas pelo simulador.

As informações exibidas na tela são persistidas na pasta `/dados`, em um arquivo XML denominado `sensores.xml`. O arquivo pode ser inspecionado diretamente no navegador pelo endereço `http://localhost/mvc_simulador/dados/sensores.xml`, conforme ilustrado na Figura 5.

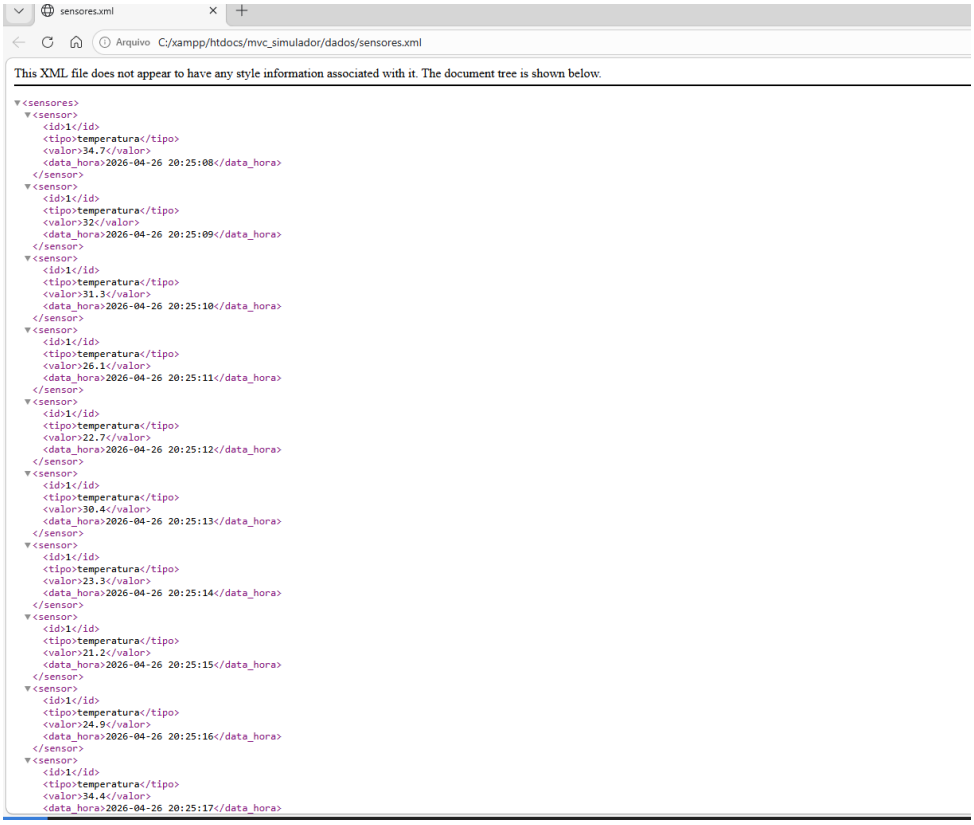

Figura 5 – Persistência das leituras de temperatura no arquivo `sensores.xml`.

5.2 Resultado de Propriedade Intelectual

A criação do backend SimuSensor culminou na formação de um ativo de Propriedade Intelectual formal. A Lei nº 9.610/1998 garante a proteção dos direitos autorais, e o registro de Programa de Computador oferece uma base legal sólida, comprovando a autoria e a originalidade da solução.

O registro formal do programa de computador SimuSensor — Arquitetura Model-View-Controller (MVC) para Inserção Automatizada de Sensores por meio de Simulação foi submetido pelo Instituto Federal Goiano - Campus Rio Verde ao Instituto Nacional da Propriedade Intelectual (INPI), resultando no Registro de Programa de Computador BR512025003342-7 (ver Anexo B), o que comprova o mérito técnico e a originalidade da solução desenvolvida.

Referências

ABIRI, R. et al. Application of digital technologies for ensuring agricultural productivity. *Heliyon*, Elsevier, v. 9, n. 11, p. e22601, 2023. Citado 2 vezes nas páginas 5 e 6.

ADITYA, B. R. et al. Design an agricultural soil and environment monitoring system based on iot. *Pertanika Journal of Science & Technology*, Universiti Putra Malaysia Press, v. 32, n. 6, p. 2575–2589, 2024. Citado 2 vezes nas páginas 5 e 6.

ALVES, G. C. et al. Monitoramento e controle de temperatura numa estufa para secagem de materiais sólidos. *Caderno Pedagógico*, v. 21, n. 7, p. e6050, jul. 2024. Disponível em: <https://ojs.studiespublicacoes.com.br/ojs/index.php/cadped/article/view/6050>. Citado na página 1.

Autodesk. *Tinkercad Circuits*. 2025. Simulador online de circuitos e Arduino. Acesso em: 10 nov. 2025. Disponível em: <https://www.tinkercad.com/>. Citado na página 1.

BRESSAN, S. et al. *Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web: VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb. Revised Papers*. Springer Berlin Heidelberg, 2003. (Lecture Notes in Computer Science). ISBN 9783540365563. Disponível em: <https://books.google.com.br/books?id=kZVuCQAAQBAJ>. Citado na página 10.

BROOKS, D. *An Introduction to PHP for Scientists and Engineers: Beyond JavaScript*. Springer London, 2009. ISBN 9781848002371. Disponível em: <https://books.google.com.br/books?id=WVTBxbvw5zQC>. Citado na página 9.

CHAOUCHI, H. (Ed.). *The Internet of Things: Connecting Objects to the Web*. London and Hoboken, NJ: ISTE Ltd and John Wiley & Sons, Inc., 2010. ISBN 978-1-84821-140-7. Citado na página 2.

CHATAUT, R.; PHOUMMALAYVANE, A.; AKL, R. Unleashing the power of IoT: A comprehensive review of IoT applications and future prospects in healthcare, agriculture, smart homes, smart cities, and industry 4.0. *Sensors*, MDPI, v. 23, p. 7194, Aug 2023. Citado na página 3.

FOUNTAS, S. et al. The future of digital agriculture: Technologies and opportunities. *IT Professional*, IEEE, v. 22, n. 1, p. 24–28, 2020. Citado na página 3.

GORALSKI, W. Chapter 26 - hypertext transfer protocol. In: GORALSKI, W. (Ed.). *The Illustrated Network (Second Edition)*. Second edition. Boston: Morgan Kaufmann, 2017. p. 661–684. ISBN 978-0-12-811027-0. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780128110270000266>. Citado na página 9.

GOURLEY, D.; TOTTY, B. *HTTP: The definitive guide*. Sebastopol, CA: O'Reilly Media, Inc., 2002. Citado na página 10.

ISSA, A. A. et al. Farming in the digital age: Smart agriculture with ai and iot. In: EDP SCIENCES. *E3S Web of Conferences*. [S.l.], 2024. v. 477, p. 00081. Citado na página 5.

McKinsey Global Institute. *The Internet of Things: Catching up to an accelerating opportunity*. [S.l.], 2021. Report. Disponível em: <https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-internet-of-things-catching-up-to-an-accelerating-opportunity>. Citado na página 2.

QUEIROZ, D. M. de et al. Sensors applied to digital agriculture: A review. *Revista Ciência Agronômica*, Universidade Federal do Ceará, v. 51, n. Special Agriculture 4.0, p. e20207751, 2020. Citado 2 vezes nas páginas 5 e 6.

REIS, Â. V. d. et al. Technological trends in digital agriculture and their impact on agricultural machinery development practices. *Revista Ciência Agronômica*, Universidade Federal do Ceará, v. 51, n. Special Agriculture 4.0, p. e20207740, 2020. ISSN 1806-6690. Citado na página 2.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011. Tradução de Ivan Bosnic e Kalinka G. de O. Gonçalves; Revisão Técnica de Kechi Hirama. Citado na página 3.

SOTT, M. K. et al. Agriculture 4.0 and smart sensors: The scientific evolution of digital agriculture: Challenges and opportunities. *Preprints*, MDPI, 2021. Preprint. Citado 2 vezes nas páginas 5 e 6.

VERMESAN, O. et al. *Internet of Things Strategic Research Roadmap*. Belgium, 2011. Disponível em: <http://www.internet-of-things-research.eu>. Acesso em: 30 nov. 2025. Citado na página 2.

VILLAMIL, S.; HERNáNDEZ, C.; TARAZONA, G. An overview of internet of things. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 2020. Citado na página 2.

VOORHEES, D. P. *Guide to Efficient Software Design: An MVC Approach to Concepts, Structures, and Models*. 1. ed. Cham: Springer Nature Switzerland AG, 2020. (Texts in Computer Science). ISBN 978-3-030-28500-5. Citado na página 3.

Wokwi. *Wokwi Arduino Simulator*. 2025. Simulador online para Arduino e ESP32. Acesso em: 10 nov. 2025. Disponível em: <https://wokwi.com/>. Citado na página 1.

ZIKRIA, Y. B. et al. Next-generation internet of things (iot): Opportunities, challenges, and solutions. *Sensors*, v. 21, n. 4, 2021. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/21/4/1174>. Citado na página 2.

Anexos

ANEXO A – Ficha-catalográfica

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema Integrado de Bibliotecas do IF Goiano - SIBi

Alves de Jesus Junior, Ivan

J95 Inserção automatizada de sensores por meio de simulação em uma Arquitetura model-View-Controller (MVC) / Ivan Alves de Jesus Junior. Rio verde 2026.

26f. il.

Orientador: Prof. Dr. Adriano Soares de Oliveira Bailão.

Tcc (Bacharel) - Instituto Federal Goiano, curso de 0219201 - Bacharelado em Ciência da Computação - Integral - Rio Verde (Campus Rio Verde).

I. Título.

ANEXO B – Comprovante de Registro de Programa de Computador

REPÚBLICA FEDERATIVA DO BRASIL
MINISTÉRIO DO DESENVOLVIMENTO, INDÚSTRIA, COMÉRCIO E SERVIÇOS
INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL
DIRETORIA DE PATENTES, PROGRAMAS DE COMPUTADOR E TOPOGRAFIAS DE CIRCUITOS

Certificado de Registro de Programa de Computador

Processo Nº: **BR512025003342-7**

O Instituto Nacional da Propriedade Industrial expede o presente certificado de registro de programa de computador, válido por 50 anos a partir de 1° de janeiro subsequente à data de 30/06/2025, em conformidade com o §2°, art. 2° da Lei 9.609, de 19 de Fevereiro de 1998.

Título: SimuSensor - Arquitetura Model-View-Controller (MVC) para Inserção Automatizada de Sensores por meio de Simulação

Data de publicação: 30/06/2025

Data de criação: 23/05/2025

Titular(es): INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA GOIANO

Autor(es): ADRIANO SOARES DE OLIVEIRA BAILÃO; CAÍKE DA ROCHA DAMKE; IVAN ALVES DE JESUS JUNIOR; ANDREA BARBOZA PROTO SARDI

Linguagem: PHP

Campo de aplicação: IF-02

Tipo de programa: SM-01

Algoritmo hash: SHA-256

Resumo digital hash: c195905e157c4d841c11de67ac5223d4a26c5753cc28d7c537bee7a691a0201b

Expedido em: 29/07/2025

Aprovado por:
Carlos Alexandre Fernandes Silva
Chefe da DIPTO