LUMEN ET VIRTUS
ISSN: 2177-2789

SISTEMA WEB DE GERENCIAMENTO DE DISPOSITIVOS IOT

WEB IOT DEVICE MANAGEMENT SYSTEM

SISTEMA DE GESTIÓN DE DISPOSITIVOS WEB IOT

https://doi.org/10.56238/levv16n53-026

Data de submissão: 06/09/2025 **Data de publicação:** 06/10/2025

Ângelo Gabriel Holandini Freitas
Tecnólogo em Análise e Desenvolvimento de Sistemas
Instituição: Instituto Federal de São Paulo (IFSP), Campinas
E-mail: angelo.freitas@aluno.ifsp.edu.br

Glauber da Rocha Balthazar
Doutor em Engenharia de Sistemas Agrícolas
Instituição: Instituto Federal de São Paulo (IFSP), Campinas
E-mail: glauber.balthazar@ifsp.edu.br
Orcid: https://orcid.org/0000-0002-1993-6621
Lattes: http://lattes.cnpq.br/1724935313124948

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema web para gerenciamento de dispositivos IoT, com foco na coleta, transmissão, armazenamento e visualização de dados. Utilizou-se o microcontrolador ESP32 conectado ao sensor DHT22 para a leitura de temperatura e umidade, enviando os dados via protocolo HTTP para um servidor backend desenvolvido em Java. Os dados são armazenados em um banco de dados MySQL, administrados por um webservice construído com Java e disponibilizados através de uma aplicação web construída com React.js. A metodologia Scrum foi adotada para o gerenciamento do projeto, garantindo entregas incrementais e feedback contínuo. Foram realizados testes práticos comprovando a eficácia do sistema em ambientes controlados, validando sua capacidade de monitoramento em tempo real e geração de relatórios históricos. O sistema proposto surge como uma alternativa acessível e customizável frente a soluções comerciais existentes.

Palavras-chave: Internet das Coisas. Gerenciamento IoT. Web IoT. Dispositivos IoT na Web.

ABSTRACT

This work presents the development of a web system for managing IoT devices, with a focus on data collection, transmission, storage and visualization. The ESP32 microcontroller connected to the DHT22 sensor is used to read temperature and humidity, sending the data via HTTP protocol to a backend server developed in Java. The data is stored in a MySQL database, managed by a webservice built with Java and made available through a web application built with React.js. The Scrum methodology was adopted for project management, guaranteeing increased deliveries and continuous feedback. Foram carried out practical tests verifying the effectiveness of the system in controlled environments, validating its capacity for monitoring in real time and generating historical reports. The

proposed system emerges as an affordable and customizable alternative to existing commercial solutions.

Keywords: Internet of Things. IoT Management. Web IoT. IoT Devices on the Web.

RESUMEN

Este trabajo presenta el desarrollo de un sistema web para gestión de dispositivos IoT, con foco en la cola, transmisión, armamento y visualización de datos. Se utiliza el microcontrolador ESP32 conectado al sensor DHT22 para temperatura y humedad, enviando los datos a través del protocolo HTTP a un servidor backend desarrollado en Java. Los datos están armados en un banco de datos MySQL, administrados por un servicio web construido con Java y disponibles a través de una aplicación web construida con React.js. La metodología Scrum fue adoptada para la gestión del proyecto, garantizando entregas incrementales y retroalimentación continua. Foram realizado pruebas prácticas comprovando a eficácia do sistema em ambientes controlados, validando sua capacidade de monitoramento em tempo real e geração de relatórios históricos. El sistema propone una alternativa accesible y personalizada frente a las soluciones comerciales existentes.

Palabras clave: Internet das Coisas. Gestión IoT. Internet de las Cosas (IoT) Web. Dispositivos IoT en la Web.

1 INTRODUÇÃO

Os dispositivos de Internet das Coisas (IoT) têm se tornado fundamentais em diversos setores pois possibilitam a coleta e análise de dados em tempo real resultando em melhorias significativas na eficiência operacional e na tomada de decisões (Santos; Silva, 2023). O crescimento recente do mercado de IoT apresenta previsões indicando que o mercado de gerenciamento de dados IoT deve crescer a uma taxa composta de crescimento anual (Taxa de Crescimento Anual Composta - CAGR) de 16,58% entre 2021 e 2026 impulsionado pela modernização das arquiteturas de dados (Fernandez, 2024) e pela crescente demanda por soluções de segurança e análise de dados (Lara; Reis; Tissot-Lara; Silva, 2021). Além disso, a Cisco estima que a IoT gerará cerca de 507,5 zettabytes de dados representando um desafio considerável para as organizações em termos de armazenamento, triagem e análise (Rosa; De Souza; Da Silva, 2020).

O surgimento de novos dispositivos associado a formas facilitadoras de programação e manipulação de dados facilitaram a coleta, transmissão e digitalização de processos físicos que são utilizados em diversas etapas de gestão de serviços, como por exemplo o monitoramento meteorológico (Santos; Silva, 2023). Além disso, a computação em nuvem facilita o acesso ágil e rápido aos recursos obtidos por dispositivos de IoT (parâmetros de sensores e atuadores) possibilitando a criação de modelos computacionais que permitem o acesso sob demanda a um conjunto de recursos do sensoriamento remoto compartilhados em diferentes ambientes físicos (Nacional Institute of Standards and Technology, 2011). Portanto, a implementação de estratégias robustas de gerenciamento de dados é essencial para que as empresas possam extrair valor dos insights gerados por esses dispositivos, garantindo a segurança e a integridade das informações.

Com o avanço das tecnologias de IoT tornou-se possível desenvolver sistemas de baixo custo e alta eficiência para monitorar parâmetros em tempo real (Soares; Faria, 2021). Diversas plataformas online atualmente fazem o armazenamento de dados de IoT e monitoramento online como por exemplo ThingSpeak1, Ubidots2 e ThingsBoard3, porém todas pagas ou com planos muito limitados para acesso livre. Essas plataformas permitem a coleta contínua de dados de sensores integrados em diversos dispositivos, facilitando a análise e a tomada de decisões informadas. Além disso, a conectividade proporcionada pela IoT possibilita a integração de diferentes dispositivos e plataformas, resultando em soluções inteligentes que podem otimizar processos em setores como agricultura, saúde, manufatura e transporte. A capacidade de identificar problemas rapidamente e implementar ações corretivas em tempo real não só melhora a eficiência operacional mas também contribuem para a redução de custos e o aumento da sustentabilidade. Este trabalho apresenta a hipótese de ser possível a criação de uma solução informatizada e integrada a diferentes tipos de dispositivos de IoT para coleta, transmissão, armazenamento e disponibilização de dados em uma aplicação web possibilitando o gerenciamento dos dados de forma remota e persistente.

Desta forma, o monitoramento por meio de Internet das Coisas (IoT) tem revolucionado a forma como os dados são coletados e geridos em diversas aplicações. A transmissão de dados em tempo real, possibilitada por dispositivos conectados, permite a coleta contínua de informações cruciais para a tomada de decisões (Gubbi; Buyya; Marusic; Palaniswami, 2013). Após a coleta, o armazenamento desses dados deve ser realizado com segurança objetivando a garantia de que as informações possam ser analisadas posteriormente e, nestes casos, as soluções em nuvem têm se mostrado eficazes devido à sua escalabilidade e acessibilidade (Chang; Srirama; Buyya, 2019). A visualização adequada dos dados é igualmente importante, pois transforma informações complexas em representações compreensíveis, facilitando a interpretação e análise (Chang; Srirama; Buyya, 2019). Por fim, o gerenciamento desses dados, que envolve a persistência, organização, análise e segurança, é fundamental para extrair insights valiosos e garantir a integridade das informações (Kirk, 2012).

A utilização de tecnologias modernas, como o ESP32, e protocolos de comunicação, como Hypertext Transfer Protocol (HTTP), oferecem uma alternativa eficiente e de baixo custo para o monitoramento em tempo real. Além disso, a integração destas tecnologias com uma aplicação web permite o acesso remoto aos dados facilitando a tomada de decisão de forma rápida e precisa. Este trabalho busca preencher essa lacuna (entre coleta, armazenamento e disponibilização de dados) propondo uma solução acessível e de fácil utilização para o monitoramento de diferentes módulos sensores em ambientes remotos.

Por fim, como técnicas de implementação tecnológica dessa solução são propostos a utilização de tecnologias modernas de gerenciamento e desenvolvimento de sistemas web. Para o gerenciamento das etapas de desenvolvimento do sistema web foi adotado o Scrum como metodologia ágil de gerenciamento de projetos e como tecnologias de programação web React.js, HTML e CSS. O Scrum justifica-se por ser um framework que consegue entregar valor ao cliente de forma rápida e contínua por meio da criação de produtos de software homologáveis a cada iteração (sprint) de desenvolvimento possibilitando rápido feedback e melhoria contínua no produto final (Leffingwell, 2011). React.js justifica-se pela eficiência e popularidade dessas tecnologias no desenvolvimento de sistemas IoT e aplicações web modernas.

2 OBJETIVOS

Desenvolver uma solução web integrada para o armazenamento e gerenciamento dos dados de dispositivos IoT responsáveis pela coleta e transmissão de dados a partir de sensores e atuadores de monitoramento local.

2.1 OBJETIVOS ESPECÍFICOS

- Implementar a coleta de dados de temperatura e umidade utilizando o ESP32 e o sensor DHT22;
- Desenvolver um web service para armazenamento e disponibilização dos dados coletados;
- Criar uma aplicação web em React.js para visualização dos dados em tempo real e geração de relatórios históricos; e
- Testar o sistema em um ambiente controlado para validar sua funcionalidade e eficiência.

3 METODOLOGIA

A proposta de desenvolvimento de software foi amparada pelo suporte do framework ágil para gerenciamento de projetos Scrum. Esse framework se baseia em ciclos curtos de trabalho chamados Sprints, onde equipes auto-organizadas desenvolvem funcionalidades priorizadas, garantindo flexibilidade e adaptação a mudanças. Para tanto o trabalho foi proposto na adaptação de um ciclo iterativo e incremental de desenvolvimento de software baseado nas seguintes atividades:

- desenvolvimento do product backlog: baseado em um conjunto de requisitos funcionais e não funcionais;
- Sprint Planning: detalhamento das atividades dos sprints baseado em um modelo de vida útil estipulado no perído de uma a quatro semanas;
- Sprint Goal: determinação das metas a serem produzidas em formato de software para contemplar o modelo iterativo e incremental e
- Daily Scrum: com as reuniões de orientação do desenvolvimento do software e também de apresentação, validação e homologação dos sprints goals.

Além disso, para atender aos requisitos mínimos de documentação de software foram propostos o desenvolvimento dos seguintes diagramas:

- Casos de Uso: contemplando as funcionalidades do software;
- Componentes: contemplando as interações entre os sistemas; e
- Tabelas e Relacionamentos: contemplando a persistência dos dados.

Por fim, o software produzido foi testado utilizando uma plataforma de um microcontrolador com captura de dados de um microambiente (dados ambientais de temperatura e umidade) para posterior envio, armazenamento e visualização de dados.

4 RESULTADOS

4.1 FRAMEWORK SCRUM: PRODUCT BACKLOG

O Product Backlog é uma lista dinâmica de funcionalidades e características desejadas para o sistema de gerenciamento de dispositivos IoT. Este backlog inclui requisitos funcionais e não funcionais que guiam o desenvolvimento do projeto.

Os requisitos funcionais foram classificados como sendo todas as necessidades, funcionalidades ou características que são esperadas no software desenvolvido. Portanto, os quadros de 1 a 5 listam os requisitos funcionais do sistema, contendo seu identificador, nome, data de criação, data da última criação, prioridade, versão, e descrição.

Quadro 1 - Coleta de Dados de Sensores

Identificador	RF001
Nome	Coleta de Dados de Sensores
Data de criação	01/03/2025
Data da última alteração	N/A
Prioridade	Alta
Versão	1.0
Descrição	O sistema deve coletar dados de temperatura e umidade do sensor DHT22.

Fonte: Elaborado pelo autor (2025)

Quadro 2 - Transmissão de Dados

Identificador	RF002
Nome	Transmissão de Dados
Data de criação	01/03/2025
Data da última alteração	N/A
Prioridade	Alta
Versão	1.0
Descrição	Transmitir os dados coletados para o servidor usando protocolo HTTP.

Fonte: Elaborado pelo autor (2025)

Quadro 3 - Armazenamento de Dados

Identificador	RF003
Nome	Armazenamento de Dados
Data de criação	01/03/2025
Data da última alteração	N/A
Prioridade	Alta
Versão	1.0
Descrição	Implementar banco de dados MySQL para armazenar dados de sensores.

Fonte: Elaborado pelo autor (2025)

Quadro 4 - Visualização em Tempo Real

Identificador	RF004
Nome	Visualização em Tempo Real
Data de criação	01/03/2025
Data da última alteração	N/A
Prioridade	Média
Versão	1.0
Descrição	Exibir dados de temperatura e umidade em tempo real via aplicação web.

Fonte: Elaborado pelo autor (2025)

Quadro 5 - Geração de Relatórios

Identificador	RF005
Nome	Geração de Relatórios
Data de criação	01/03/2025
Data da última alteração	N/A
Prioridade	Média
Versão	1.0
Descrição	Permitir geração de relatórios históricos dos dados coletados.

Fonte: Elaborado pelo autor (2025)

Os requisitos não funcionais descrevem uma funcionalidade, necessidade ou característica que o sistema deve conter, ou seja, uma descrição de como o sistema realizará determinada função. Desta forma, nos quadros de 6 até 9 estão listados os requisitos não funcionais do sistema, contendo seu identificador, nome, data de criação, data da última criação, prioridade, versão, e descrição.

Quadro 6 - Segurança de Dados

Identificador	RNF001
Nome	Segurança de Dados
Data de criação	02/03/2025
Data da última alteração	N/A
Prioridade	Alta
Versão	1.0
Descrição	Garantir transmissão e armazenamento seguro dos dados.

Fonte: Elaborado pelo autor (2025)

Quadro 7 - Desempenho

Identificador	RNF002
Nome	Desempenho
Data de criação	02/03/2025
Data da última alteração	N/A
Prioridade	Alta
Versão	1.0
Descrição	Otimizar a aplicação para garantir tempos de resposta rápidos.

Fonte: Elaborado pelo autor (2025)

Quadro 8 - Usabilidade

Identificador	RNF003
Nome	Usabilidade
Data de criação	02/03/2025
Data da última alteração	N/A
Prioridade	Média
Versão	1.0
Descrição	Interface intuitiva e fácil de usar para uma experiência agradável.

Fonte: Elaborado pelo autor (2025)

Quadro 9 - Compatibilidade

Identificador	RNF004
Nome	Compatibilidade
Data de criação	02/03/2025
Data da última alteração	N/A
Prioridade	Média
Versão	1.0
Descrição	Compatibilidade do sistema com diferentes navegadores e dispositivos.

Fonte: Elaborado pelo autor (2025)

4.2 FRAMEWORK SCRUM: SPRINT PLANNING

O desenvolvimento de projeto é pensado para ser dividido em sprints - períodos de tempo onde um ou mais requisitos (funcionais ou não funcionais) são escolhidos para serem construídos e entregues. O período ideal de duração para cada sprint é de 1 a 4 semanas. Antes do início de cada sprint, é feito um planejamento de sprint (Sprint Planning, em inglês), onde é definido quantos e quais requisitos podem ser construídos e entregues. Após o término de uma sprint, é esperada a entrega de uma nova adição ao software focada nos requisitos desenvolvidos. A seguir são apresentadas a divisão dos requisitos em sprints e o tempo estimado para cada um:

- Sprint 1: Coleta e Transmissão de Dados
  - Requisitos: [RF001] Coleta de Dados de Sensores; [RF002] Transmissão de Dados
  - Duração: 2 semanas
- Sprint 2: Armazenamento
  - Requisitos: [RF003] Armazenamento de Dados
  - Duração: 2 semanas
- Sprint 3: Visualização e Relatórios
  - Requisitos: [RF004] Visualização em Tempo Real; [RF005] Geração de Relatórios
  - Duração: 4 semanas
- Sprint 4: Requisitos Não Funcionais
  - Requisitos: [RNF001] Segurança de Dados; [RNF002] Desempenho; [RNF003] Usabilidade; [RNF004] Compatibilidade
  - Duração: 4 semanas

4.3 FRAMEWORK SCRUM: SPRINT GOAL

O Sprint Goal é uma declaração de propósito para cada sprint, orientando o que precisa ser alcançado durante o ciclo de desenvolvimento. Essa meta deve alinhar os esforços do desenvolvedor e servir como um critério para avaliar o sucesso do sprint. Os objetivos devem ser revisados e ajustados conforme necessário, com base em feedback e resultados dos sprints anteriores. A seguir, são listados os sprints e seus respectivos objetivos:

- Sprint 1 Goal: Estabelecer Conectividade e Coleta de Dados
  - Objetivo: Garantir que o ESP32 esteja coletando dados de temperatura e umidade com sucesso e transmitindo-os ao servidor via protocolo HTTP. A meta é ter um fluxo de dados contínuo e confiável entre o dispositivo e o servidor.
- Sprint 2 Goal: Implementar Segurança e Persistência de Dados
  - Objetivo: Configurar o armazenamento seguro dos dados no banco de dados MySQL, garantindo que apenas usuários autenticados tenham acesso às informações. A meta é garantir a integridade e segurança dos dados armazenados.
- Sprint 3 Goal: Prover Visualização Eficaz e Relatórios Precisos
  - Objetivo: Desenvolver uma interface web que permita aos usuários visualizar dados em tempo real e acessar relatórios históricos. A meta é garantir que os dados sejam exibidos claramente e que os relatórios sejam gerados com precisão.
- Sprint 4 Goal: Assegurar Qualidade de Serviço e Compatibilidade
  - Objetivo: Otimizar o sistema para garantir segurança, escalabilidade, desempenho e compatibilidade com diferentes dispositivos e navegadores. A meta é assegurar que o sistema atenda aos requisitos não funcionais, proporcionando uma experiência de usuário consistente e eficiente.

4.4 FRAMEWORK SCRUM: DAILY SCRUM

As reuniões de Daily Scrum são fundamentais para garantir a comunicação eficaz entre o desenvolvedor e o orientador, permitindo ajustes rápidos e alinhamento sobre o progresso do projeto. No contexto deste trabalho, as reuniões com o orientador desempenham um papel similar, fornecendo feedback e orientação contínua. A seguir é apresentado o Quadro 10 (Sprint Planning) que apresenta as datas das reuniões com o orientador para validação e homologação dos Goals.

Quadro 10 - Sprint Planning e Daily Scrums

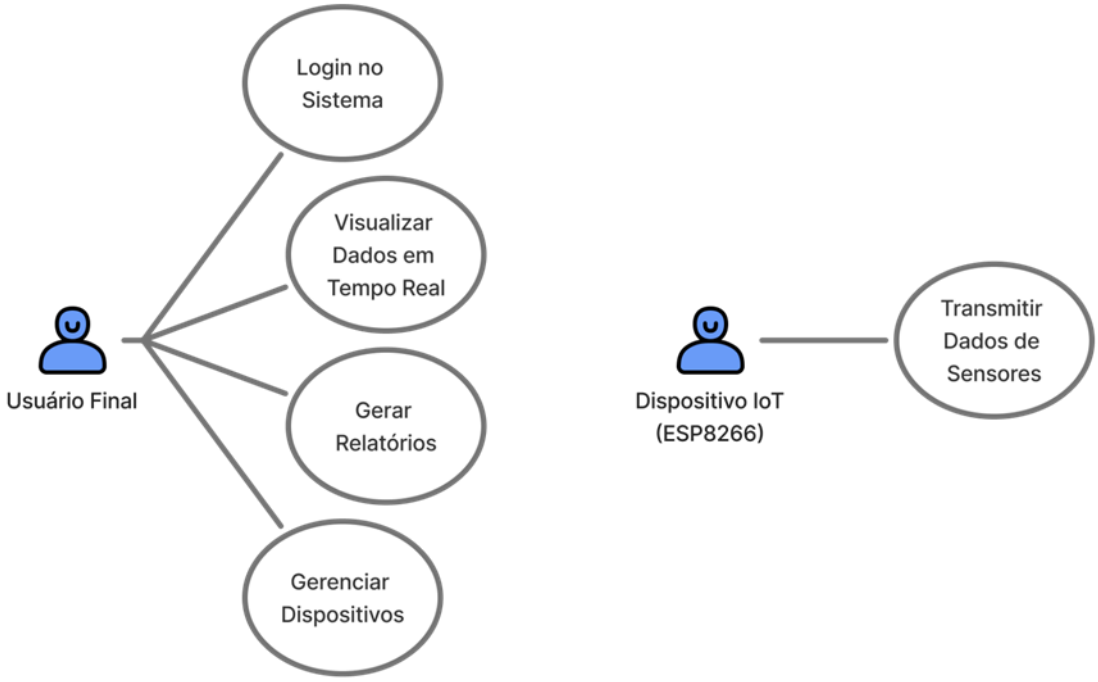
Semana	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Daily Scrums
1					07/03
2					
3					14/03
4					
5					25/03
6					
7					14/04
8					
9					
10					30/04
11					
12					

Fonte: Elaborado pelo autor (2025)

4.5 DIAGRAMAÇÕES: CASOS DE USO

Os casos de uso descrevem as interações entre atores (usuários ou sistemas externos) e o sistema com o objetivo de atingir um objetivo específico auxiliando a compreensão de como o sistema deve se comportar em diferentes situações, identificando funcionalidades, regras de negócio e possíveis fluxos de execução. Para este projeto foram determinados dois atores (usuário final e dispositivo IoT) e cinco casos de uso, todos descritos na Figura 1.

Figura 1 - Diagrama de Caso de Uso



Fonte: Elaborado pelo autor (2025)

A seguir, cada um dos itens do diagrama de caso de uso (Figura 1) é detalhado:

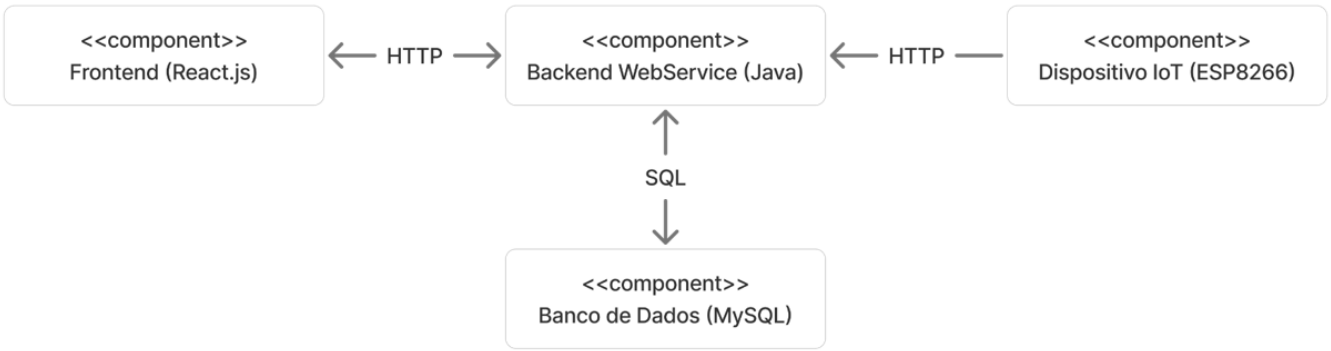
- Ator “Usuário Final”: Interage com a aplicação web para visualização de dados, geração de relatórios, e gerenciamento de dispositivos;
- Ator “Dispositivo IoT (ESP8266)”: Coleta e transmite dados de sensores;
- Caso de Uso “Login no Sistema”: Permite que o usuário se autentique no sistema;
- Caso de Uso “Visualizar Dados em Tempo Real”: Permite visualizar dados de temperatura e umidade em tempo real;
- Caso de Uso “Gerar Relatórios”: Permite ao usuário gerar relatórios sobre dados coletados ao longo do tempo;
- Caso de Uso “Configurar Dispositivos”: Permite ao usuário adicionar ou modificar configurações de dispositivos IoT;
- Caso de Uso “Gerenciar Usuários”: Permite ao usuário adicionar, remover ou modificar usuários do sistema;

- Caso de Uso “Transmitir Dados de Sensores”: Envia dados coletados para o servidor.

4.6 DIAGRAMAÇÕES: DIAGRAMA DE COMPONENTES

Esse diagrama apresenta a estrutura física e lógica de um sistema em termos de componentes, suas interfaces e dependências possibilitando a visualização e organização dos relacionamentos entre as partes modulares de um sistema de software. O diagrama é apresentado na Figura 2.

Figura 2 - Diagrama de Componentes



Fonte: Elaborado pelo autor (2025)

O detalhamento dos componentes apresentados na Figura 2 é descrito a seguir:

- Frontend (React.js)
  - Descrição: Interface de usuário para interação e visualização de dados.
  - Responsabilidades: Exibir dados em tempo real, permitir login de usuários, gerar relatórios.
- Backend WebService (Java)
  - Descrição: Processamento de lógica de negócios e comunicação com o banco de dados.
  - Responsabilidades: Autenticar usuários, processar dados recebidos dos dispositivos IoT, gerar endpoints para o frontend.
- Dispositivo IoT (ESP8266)
  - Descrição: Hardware que coleta e transmite dados de sensores.
  - Responsabilidades: Coletar dados de temperatura e umidade e transmitir para o servidor.
- Banco de Dados (MySQL)
  - Descrição: Armazenamento de dados de sensores e informações de usuários.
  - Responsabilidades: Persistir dados de temperatura e umidade, armazenar dados de autenticação.

Para que alguns dos componentes sejam executados e se comuniquem entre si, foi utilizada a ferramenta Jelastic, um serviço em nuvem que fornece plataforma multinuvem baseada em tecnologia de contêiner. Nela, é possível criar contêineres de diversas tecnologias e disponibilizá-las para utilização em nuvem. Com o Jelastic, foram criados os seguintes ambientes e contêineres:

- Ambiente **webapp-iot**

- Contêiner **Node.js** - utilizado para hospedar o componente Frontend (React.js).
- Ambiente **webservice-db-iot**
- Contêiner **Apache** - utilizado para hospedar o componente Backend WebService (Java);
- Contêiner **MySQL** - utilizado para hospedar o componente Banco de Dados (MySQL);
- Contêiner **Maven** - utilizado para executar a *build* do componente Backend WebService (Java).

A Figura 3 apresenta a tabela de ambientes e contêineres criados no Jelastic.

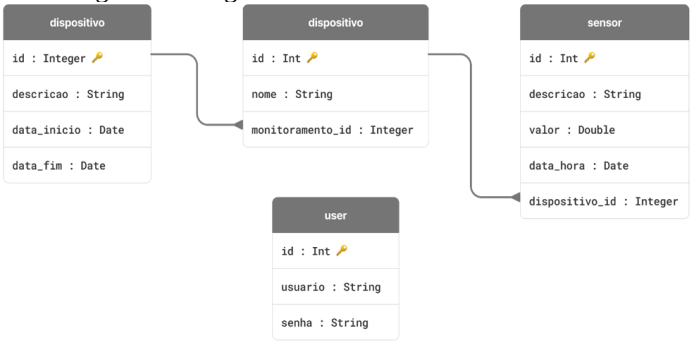
Figura 3 - Ambientes e contêineres criados no Jelastic

Fonte: Elaborado pelo autor (2025)

4.7 DIAGRAMAÇÕES: DIAGRAMA DE TABELAS E RELACIONAMENTOS (DTR)

Trata-se de uma representação visual que descreve a estrutura do banco de dados relacional utilizado neste projeto (incluindo suas tabelas, atributos e relacionamentos) objetivando apresentar o planejamento da documentação do banco de dados para posterior implementação. A Figura 4 apresenta o DTR.

Figura 4 - Diagrama de Tabelas e Relacionamentos



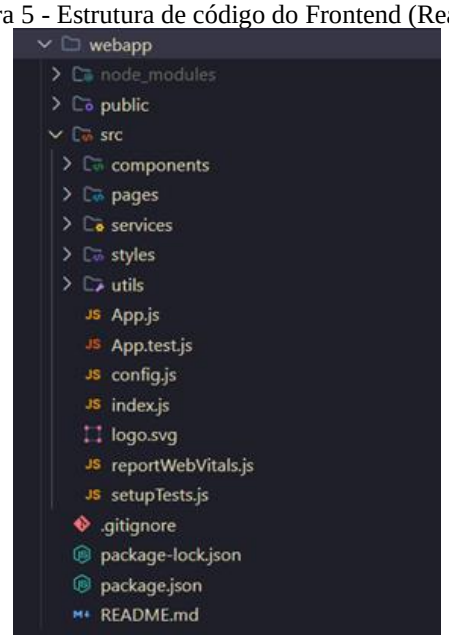
Fonte: Elaborado pelo autor (2025)

4.8 PROGRAMAÇÃO E TESTES: CODIFICAÇÃO

A seguir são apresentados os principais trechos de código fonte (mais representativos) desenvolvidos durante o projeto, ilustrando a estruturação lógica e organização adotada na implementação do sistema, onde é possível visualizar como o código foi organizado próximo ao padrão arquitetural MVC (Model-View-Controller), evidenciando a separação entre as camadas de interface, lógica de negócios e persistência dos dados. Além disso, são destacadas partes do código responsáveis pela comunicação entre o dispositivo IoT e o sistema central, bem como a rotina responsável por receber, tratar e armazenar os dados coletados pelos sensores.

A Figura 5 apresenta a estrutura de código do Frontend (React.js), que segue um padrão semelhante ao padrão arquitetural MVC, onde as pages fazem o papel de exibição dos dados, e os services fazem o papel de controladores de processamento entre os demais componentes.

Figura 5 - Estrutura de código do Frontend (React.js)



Fonte: Elaborado pelo autor (2025)

A Figura 6 apresenta o código do componente principal (App) do Frontend, onde são definidos a barra de navegação e as rotas de acesso da aplicação.

Figura 6 - Código inicial do frontend

```jsx
import React from 'react'
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Navbar from './components/Navbar'
import PrivateRoute from './components/PrivateRoute';
import Dashboard from './pages/Dashboard';
import Sensores from './pages/Sensores';
import './styles/App.css';
import Login from './components/Login';

function App() {
    return (
        <Router>
            <div className='App'>
                <Navbar />
                <div className='App-body'>
                    <Routes>
                        <Route path='/' element={
                            <PrivateRoute element={<Dashboard />} />
                        }/>
                        <Route path='/sensores' element={
                          <PrivateRoute element={<Sensores />} />
                        }/>
                        <Route path='/login' element={<Login />} />
                    </Routes>
                </div>
            </div>
        </Router>
    )
}

export default App;
```

Fonte: Elaborado pelo autor (2025)

A Figura 7 apresenta o código da página Dashboard, responsável pela exibição dos gráficos de valores de temperatura e umidade capturados pelos sensores DHT22.

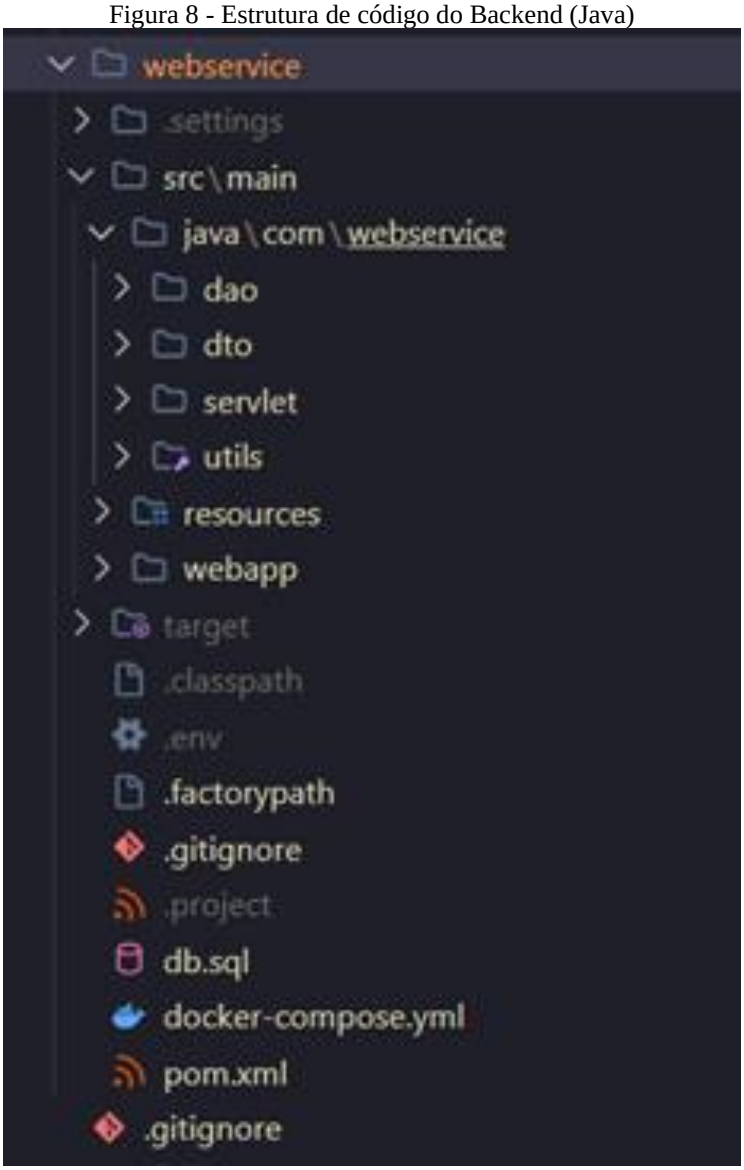
Figura 7 - Código da página Dashboard

```jsx
const Dashboard = () => {

    const [sensorData, setSensorData] = useState([]);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);

    useEffect(() => {
        const fetchSensores = async () => {
            try {
                const data = await getAllSensores();
                setSensorData(data);
            } catch (error) {
                setError('Failed to fetch sensores: ' + error);
            } finally {
                setLoading(false);
            }
        }

        fetchSensores();
    }, [])

    if (loading) return <div>Carregando...</div>
    if (error) return <div>{error}</div>

    return (
        <div>
            <h1>Dashboards</h1>
            <SensorChart sensorData={sensorData} />
        </div>
    )
}

export default Dashboard;
```

Fonte: Elaborado pelo autor (2025)

A Figura 8 apresenta a estrutura de código do Backend (Java), que segue um padrão semelhante ao padrão arquitetural MVC, onde os DAO (Data Access Object) fazem o papel dos modelos de dados, os DTO (Data Transfer Object) fazem o papel de controladores de processamento entre os demais componentes e os *servlets* fazem o papel de comunicação com o Banco de Dados (MySQL).

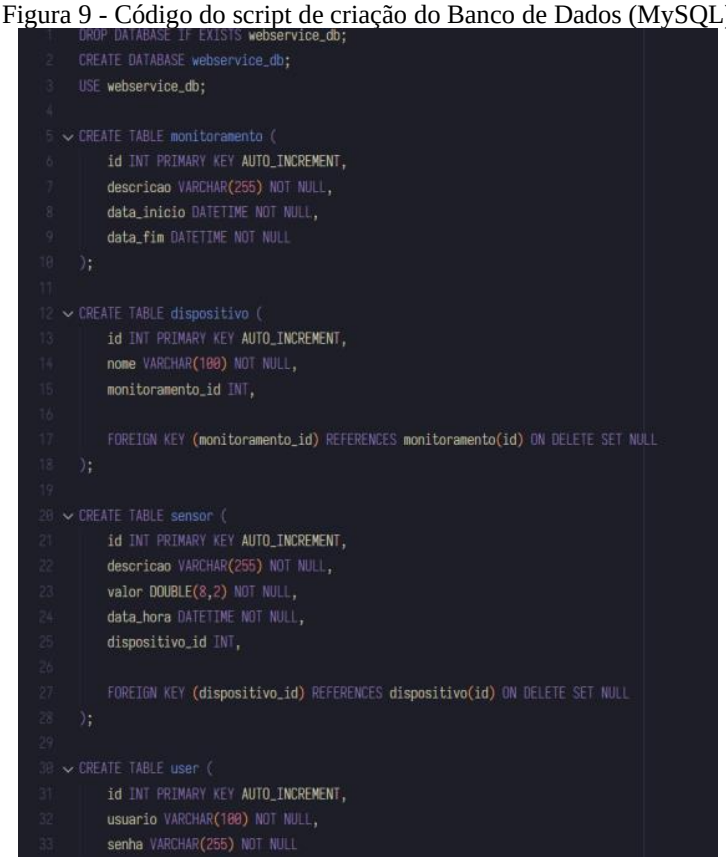
Figura 8 - Estrutura de código do Backend (Java)



Fonte: Elaborado pelo autor (2025)

A Figura 9 apresenta o código do script de criação e população do Banco de Dados (MySQL), onde são criadas as tabelas Monitoramento, Dispositivo, Sensor e User.

Figura 9 - Código do script de criação do Banco de Dados (MySQL)

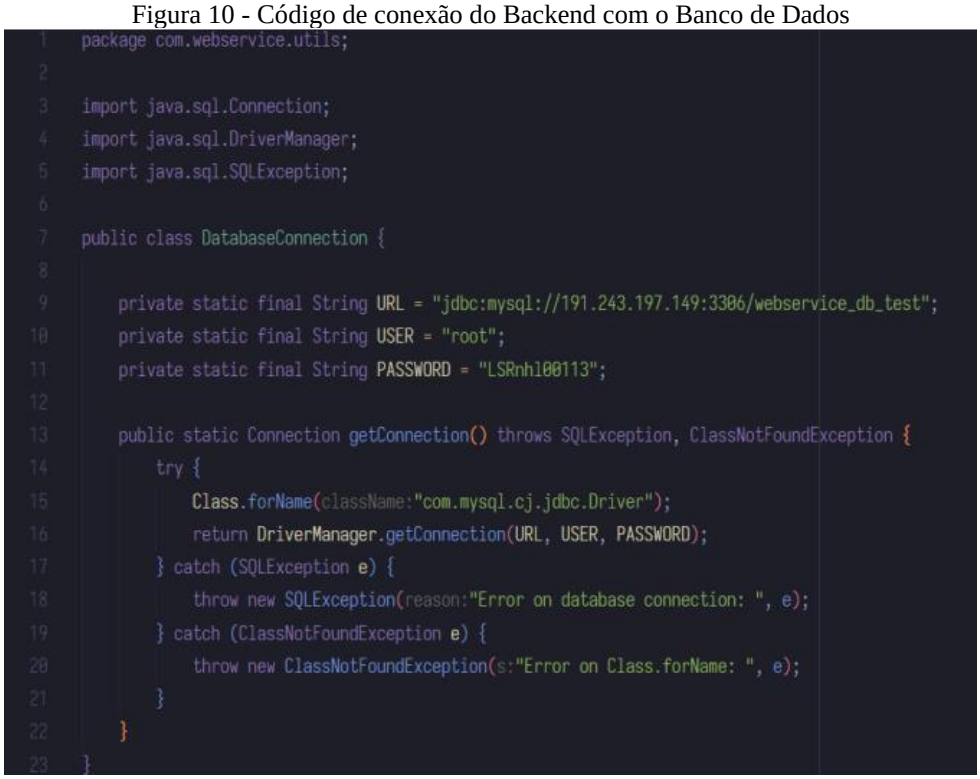


```sql
DROP DATABASE IF EXISTS webservice_db;
CREATE DATABASE webservice_db;
USE webservice_db;

CREATE TABLE monitoramento (
    id INT PRIMARY KEY AUTO_INCREMENT,
    descricao VARCHAR(255) NOT NULL,
    data_inicio DATETIME NOT NULL,
    data_fim DATETIME NOT NULL
);

CREATE TABLE dispositivo (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,
    monitoramento_id INT,

    FOREIGN KEY (monitoramento_id) REFERENCES monitoramento(id) ON DELETE SET NULL
);

CREATE TABLE sensor (
    id INT PRIMARY KEY AUTO_INCREMENT,
    descricao VARCHAR(255) NOT NULL,
    valor DOUBLE(8,2) NOT NULL,
    data_hora DATETIME NOT NULL,
    dispositivo_id INT,

    FOREIGN KEY (dispositivo_id) REFERENCES dispositivo(id) ON DELETE SET NULL
);

CREATE TABLE user (
    id INT PRIMARY KEY AUTO_INCREMENT,
    usuario VARCHAR(100) NOT NULL,
    senha VARCHAR(255) NOT NULL
```

Fonte: Elaborado pelo autor (2025)

A Figura 10 apresenta o código da classe responsável pela conexão do Backend (Java) com o Banco de Dados (MySQL).

Figura 10 - Código de conexão do Backend com o Banco de Dados

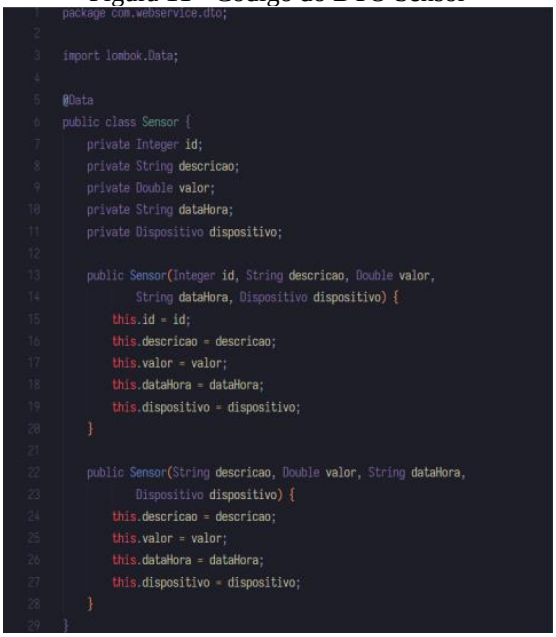


```java
package com.webservice.utils;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {

    private static final String URL = "jdbc:mysql://191.243.197.149:3306/webservice_db_test";
    private static final String USER = "root";
    private static final String PASSWORD = "LSRnhl00113";

    public static Connection getConnection() throws SQLException, ClassNotFoundException {
        try {
            Class.forName(className:"com.mysql.cj.jdbc.Driver");
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (SQLException e) {
            throw new SQLException(reason:"Error on database connection: ", e);
        } catch (ClassNotFoundException e) {
            throw new ClassNotFoundException(s:"Error on Class.forName: ", e);
        }
    }
}
```

Fonte: Elaborado pelo autor (2025)

A Figura 11 apresenta o código do modelo DTO Sensor (mesma função de Model), responsável por guardar os atributos necessários para registrar os dados de sensores. O mesmo ocorre para os DTO Monitoramento, Dispositivo e User.

Figura 11 - Código do DTO Sensor



```java
package com.webservice.dto;

import lombok.Data;

@Data
public class Sensor {
    private Integer id;
    private String descricao;
    private Double valor;
    private String dataHora;
    private Dispositivo dispositivo;

    public Sensor(Integer id, String descricao, Double valor,
            String dataHora, Dispositivo dispositivo) {
        this.id = id;
        this.descricao = descricao;
        this.valor = valor;
        this.dataHora = dataHora;
        this.dispositivo = dispositivo;
    }

    public Sensor(String descricao, Double valor, String dataHora,
            Dispositivo dispositivo) {
        this.descricao = descricao;
        this.valor = valor;
        this.dataHora = dataHora;
        this.dispositivo = dispositivo;
    }
}
```

Fonte: Elaborado pelo autor (2025)

A Figura 12 apresenta o código do servlet Serviços, responsável por administrar a comunicação do Dispositivo ESP8266 com os DTO (modelos) do Backend Webservice (Java).

Figura 12 - Código do servlet Serviços

```java
public class ServicosServlet extends HttpServlet {

    private MonitoramentoDAO monitoramentoDAO;
    private DispositivoDAO dispositivoDAO;
    private SensorDAO sensorDAO;
    private Gson gson;

    private final String[] valoresNotSensor = {
        "monitoramento",
        "dispositivo",
        "datahora"
    };

    @Override
    public void init() throws ServletException {···

    protected void doGet(HttpServletRequest request, HttpServletResponse response)···

    protected void doPost(HttpServletRequest request, HttpServletResponse response)···

    protected void execute(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        try {
            response.setContentType(type:"application/json");
            String valores = request.getParameter(name:"valores");

            if (valores == null) {
                response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
                throw new Exception(message:"Parameter 'valores' not found.");
            }

            Map<String, String> valoresMap = new HashMap<String, String>();
```

Fonte: Elaborado pelo autor (2025)

A Figura 13 apresenta o código do servlet Sensor, responsável por administrar a comunicação entre o Frontend (React.js), o DAO e DTO (modelo) Sensor.

Figura 13 - Código do servlet Sensor

```java
@WebServlet("/api/sensores/*")
public class SensorServlet extends HttpServlet {

    private SensorDAO sensorDAO;
    private DispositivoDAO dispositivoDAO;
    private Gson gson;

    @Override
    public void init() throws ServletException {
        try {
            Connection connection = DatabaseConnection.getConnection();
            sensorDAO = new SensorDAO(connection);
            dispositivoDAO = new DispositivoDAO(connection);
            gson = new Gson();
            System.out.println("SensorServlet.init(): Database connection " +
                "established successfully.");
        } catch (SQLException | ClassNotFoundException e) {
            throw new ServletException("Database connection error on " +
                "SensorServlet.init: ", e);
        }
    }

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String pathInfo = request.getPathInfo();
        response.setContentType(type:"application/json");

        if (pathInfo != null && pathInfo.matches(regex:"/\\d+")) {
            Integer id = Integer.parseInt(pathInfo.substring(beginIndex:1));
            try {
                Sensor sensor = sensorDAO.getSensorById(id);
```

Fonte: Elaborado pelo autor (2025)

A Figura 14 apresenta o código do Dispositivo ESP8266, responsável por estabelecer a conexão com a internet, iniciar a leitura do sensor DHT22, e enviar os valores capturados pelo sensor para o Backend Webservice (Java).

Figura 14 - Código do Dispositivo ESP8266

```
// Configurações DHT22
#define DHTPIN D5
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
float temperature = 0;
float humidity = 0;

void setup() {
  Serial.begin(115200);
  //start DHT22
  dht.begin();
}

void loop() {

  if(lerDados()){
    conectarWifi();
    enviarDadosServidor();
  }else{
    Serial.println("Falha na leitura do sensor DHT22!");
  }

  // Aguarda 2 minutos antes de executar novamente
  delay(600000);
}

void conectarWifi(){
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Conectando à rede WiFi...");
  }
  Serial.println("Conectado à rede WiFi!");
}

boolean lerDados(){
   // Lê a temperatura e umidade
  temperature = dht.readTemperature();
  Serial.print("Temperatura: ");Serial.println(temperature);
  humidity = dht.readHumidity();
  Serial.print("Umidade: ");Serial.println(humidity);
  boolean retorno = true;
```

Fonte: Elaborado pelo autor (2025)

4.9 PROGRAMAÇÃO E TESTES: AVALIAÇÃO DO SOFTWARE

Com o objetivo de validar o funcionamento do sistema desenvolvido, foi realizada uma série de testes práticos envolvendo todo o fluxo desde a coleta dos dados no dispositivo IoT até a sua correta exibição no sistema web. Inicialmente, foi montado o circuito com o sensor DHT conectado ao ESP, responsável pela leitura dos dados de temperatura e umidade (registrado com uma foto feita com o celular). Em seguida, foi realizado o envio dos dados coletados ao servidor, comprovado por meio de prints do Serial Monitor do Arduino IDE. No ambiente web, foram capturadas imagens demonstrando a tabela sendo populada com os dados recebidos, além das telas do sistema web — como a tela de login e navegação — evidenciando que os dados são armazenados e visualizados corretamente pelo usuário.

A Figura 15 apresenta o circuito com o sensor DHT22 conectado ao Dispositivo ESP8266.

15 - Circuito montado com ESP8266 e DHT22)

Fonte: Elaborado pelo autor (2025)

A Figura 16 apresenta o mesmo circuito após o código do Dispositivo ESP8266 (Figura 14) ser executado.

Figura 16 - ESP8266 após ser executado

Fonte: Elaborado pelo autor (2025)

A Figura 17 apresenta a saída do ESP8266 após executar o código programado, mostrando que o sensor capturou os valores de temperatura e umidade do ambiente, e as enviou para o Backend Webservice (Java).

Figura 17 - Saída do ESP8266 no momento da conexão com o Backend (Java)

```
20:42:42.814 -> http://192.168.1.27:8181/WebService/servicos?valores=temperatura=26.20;umidade=64.00
20:42:49.852 -> Temperatura: 26.20
20:42:49.852 -> Umidade: 63.90
```

Fonte: Elaborado pelo autor (2025)

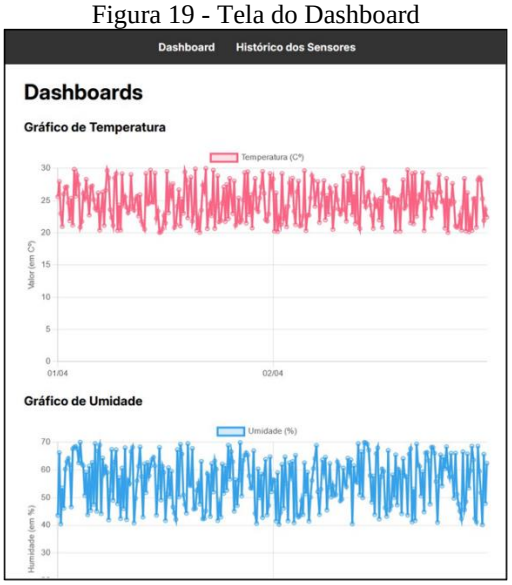
A Figura 18 apresenta a tela de Login, onde o usuário deve informar o seu usuário e sua senha para ter acesso às demais telas da aplicação.

Figura 18 - Tela de Login

Fonte: Elaborado pelo autor (2025)

A Figura 19 apresenta a tela de Dashboard, onde é possível visualizar os gráficos de valores de temperatura e umidade capturados pelos sensores DHT22.

Figura 19 - Tela do Dashboard



Fonte: Elaborado pelo autor (2025)

A Figura 20 apresenta a tela de Histórico dos Sensores, onde é possível visualizar o histórico de valores de temperatura e umidade capturados pelos sensores, além da data e hora da captura, e do Monitoramento e do Dispositivo vinculados ao sensor.

Figura 20 - Tela do Histórico dos Sensores

Monitoramento	Dispositivo	Descrição	Valor	Data e Hora
Monitoramento 01	Dispositivo 01	temperatura	25.61	2025-04-01 00:00:00
Monitoramento 01	Dispositivo 01	temperatura	27.96	2025-04-01 00:10:00
Monitoramento 01	Dispositivo 01	temperatura	22.97	2025-04-01 00:20:00
Monitoramento 01	Dispositivo 01	temperatura	20.98	2025-04-01 00:30:00
Monitoramento 01	Dispositivo 01	temperatura	26.00	2025-04-01 00:40:00
Monitoramento 01	Dispositivo 01	temperatura	27.03	2025-04-01 00:50:00
Monitoramento 01	Dispositivo 01	temperatura	27.15	2025-04-01 01:00:00
Monitoramento 01	Dispositivo 01	temperatura	24.66	2025-04-01 01:10:00
Monitoramento 01	Dispositivo 01	temperatura	21.87	2025-04-01 01:20:00
Monitoramento 01	Dispositivo 01	temperatura	25.36	2025-04-01 01:30:00
Monitoramento 01	Dispositivo 01	temperatura	21.19	2025-04-01 01:40:00
Monitoramento 01	Dispositivo 01	temperatura	29.85	2025-04-01 01:50:00
Monitoramento 01	Dispositivo 01	temperatura	25.69	2025-04-01 02:00:00
Monitoramento 01	Dispositivo 01	temperatura	28.92	2025-04-01 02:10:00

Fonte: Elaborado pelo autor (2025)

5 CONSIDERAÇÕES FINAIS

O desenvolvimento do sistema web para gerenciamento de dispositivos IoT mostrou-se tecnicamente viável e economicamente acessível, graças ao uso de tecnologias amplamente difundidas e de baixo custo. A integração entre o hardware (ESP32 e DHT22) e o software validou o fluxo completo do sistema, desde a coleta de dados até sua exibição em tempo real e armazenamento seguro. A utilização da metodologia Scrum contribuiu significativamente para a organização do projeto, permitindo flexibilidade e correções rápidas durante o desenvolvimento.

Os testes realizados comprovaram a eficiência do sistema na transmissão de dados com baixa latência e seu correto armazenamento no banco de dados MySQL. A interface desenvolvida com React.js destacou-se pela usabilidade, responsividade e compatibilidade em diferentes dispositivos e navegadores. Além disso, a funcionalidade de geração de relatórios históricos reforçou a utilidade do sistema para análises posteriores, agregando valor às informações coletadas pelos sensores.

Como sugestão para trabalhos futuros, é possível evoluir o sistema com funcionalidades como autenticação avançada, notificações em tempo real, gráficos interativos e suporte a múltiplos dispositivos IoT conectados simultaneamente. Também é promissora a implementação de regras de alerta automáticas com base em condições pré-definidas, aumentando a aplicabilidade do sistema em

ambientes industriais e residenciais. Assim, este trabalho apresenta uma solução prática, escalável e de código aberto que contribui para a disseminação do uso de tecnologias IoT em diferentes contextos.

REFERÊNCIAS

ADAMS, A.; BISCH, N.; NEWCOMER, J. **Fullstack React: The Complete Guide to ReactJS and Friends**. Fullstack.io, 2021.

ALEXANDER, M. **The React Ecosystem: Tools and Libraries for Modern Web Development**. Springer, 2022.

BALTHAZAR, Glauber da Rocha e SILVEIRA, Robson Mateus Freitas e SILVA, Iran José Oliveira da. **Use of multi-agent systems and the Internet of Things to monitor the environment of commercial broiler poultry houses through specific air enthalpy**. Journal of Animal Behaviour and Biometeorology, v. 12, p. 1-9, 2024Tradução . . Disponível em: https://doi.org/10.31893/jabb.2024012. Acesso em: 26 set. 2025.

CARVALHO, D. S., BALTHAZAR, G. R., DIAS, C. R., ARAÚJO, M. A. P., & MONTEIRO, P. H. R. (2006). **S²O: Uma Ferramenta de Apoio ao Aprendizado de Sistemas Operacionais**. In Anais do XIV Workshop sobre Educação em Computação, Campo Grande/MS.

CHANG, C.; SRIRAMA, S. N.; BUYYA, R. **Internet of Things (IoT) and New Computing Paradigms**. Fog And Edge Computing, [S.L.], p. 1-23, 11 jan. 2019. Wiley. Disponível em: http://dx.doi.org/10.1002/9781119525080.ch1. Acesso em: 9 mar. 2025.

CROCKFORD, D. **JavaScript: The Good Parts**. O'Reilly Media, 2018. Disponível em: https://www.oreilly.com/library/view/javascript-the-good/9780596517748/. Acesso em: 10 mar. 2025.

DA ROCHA BALTHAZAR, Glauber; SILVEIRA, Robson Mateus Freitas; DA SILVA, Iran Jose Oliveira. **Use of robotics in broiler production systems: a relationship between technology, environment and production**. Tropical Animal Health and Production, v. 57, n. 3, p. 1-21, 2025.

FERNANDEZ, J. **Tamanho do mercado de gerenciamento de dados IoT e análise de ações – Tendências e previsões de crescimento (2024 – 2029**). Disponível em: https://www.mordorintelligence.com/pt/industry-reports/iot-data-management-market. Acesso em: 01 abr. 2025.

FLANAGAN, D. **JavaScript: The Definitive Guide**. 7. ed. O'Reilly Media, 2020. Disponível em: https://www.oreilly.com/library/view/javascript-the-definitive/9781491952016/. Acesso em: 16 abr. 2025.

GONZÁLEZ, J. L.; MARTÍNEZ, P.; GARCÍA, F. **IoT-Based Web Systems: Challenges and Opportunities**. Journal of Internet of Things and Web Development, v. 12, n. 2, p. 45-62, 2022. Disponível em: https://www.jiotwd.com/article/iot-web-systems. Acesso em: 16 abr. 2025.

GONÇALVEZ, A., SOUZA, M., SILVA, R., SILVA, D., BUENO, P., Balthazar, G. R. (2014). **Desenvolvimento de Jogos Educacionais na Área de Matemática em Escola de Ensino Fundamental**, XIX Congresso Internacional de Informática Educativa, p. 622 627.

GOODRICH, M. T.; TAMASSIA, R.; MOUNT, D. M. **Data Structures and Algorithms in JavaScript**. Wiley, 2021. Disponível em: https://www.wiley.com/en-us/Data+Structures+and+Algorithms+in+JavaScript-p-9781119701 234. Acesso em: 16 abr. 2025.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. **Internet of Things (IoT): a vision, architectural elements, and future directions**. Future Generation Computer Systems, [S.L.], v. 29,

n. 7, p. 1645-1660, set. 2013. Elsevier BV. Disponível em: http://dx.doi.org/10.1016/j.future.2013.01.010. Acesso em: 10 abr. 2025.

KIRK, A. **Data Visualization: A Structured Design Approach to Equip You with the Knowledge of How to Successfully Accomplish Any Data Visualization Challenge Efficiently and Effectively**. Birmingham, Uk: Packt Press, 2012. 189 p. Disponível em: https://openlibrary.telkomuniversity.ac.id/pustaka/172826/data-visualization-a-successful-desi gn-process.html. Acesso em: 25 mar. 2025.

LARA, J. E.; REIS, L. J.; TISSOT-LARA, T. A.; SILVA, A. O. **Admirável mundo novo na perspectiva da tríade: internet das coisas, pessoas e mercados**. Perspectivas em Ciência da Informação, [S.L.], v. 26, n. 2, p. 124-150, jun. 2021. FapUNIFESP (SciELO). Disponível em: http://dx.doi.org/10.1590/1981-5344/3825. Acesso em: 23 mar. 2025.

LEFFINGWELL, D. **Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise (Agile Software Development Series)**. Boston, Usa: Addison-Wesley Educational Publishers Inc, 2011. p. 560. Disponível em: https://www.oreilly.com/library/view/agile-software-requirements/9780321685438/. Acesso em 23 mar. 2025.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **Guidelines on Security and Privacy in Public Cloud Computing**. 2011. Disponível em: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-144.pdf. Acesso em: 25 mar. 2025.

OLIVEIRA, A. P.; COSTA, L. F. **Integração de Sensores de Baixo Custo em Sistemas IoT para Monitoramento Ambiental**. Anais do Congresso Brasileiro de Automação e Controle, São Paulo, 2024. Disponível em: https://www.cbac2024.org. Acesso em: 16 abr. 2025.

OLIVEIRA, R. T. **Java no Desenvolvimento de Sistemas Industriais**. FAM, 2023. Disponível em: https://fam-cati2023.pdf. Acessado em: 16 abr. 2025.

ROSA, C. M.; SOUZA, P. A. R.; SILVA, J. M. **Inovação em saúde e internet das coisas (IoT): um panorama do desenvolvimento científico e tecnológico**. Perspectivas em Ciência da Informação, [S.L.], v. 25, n. 3, p. 164-181, jul. 2020. FapUNIFESP (SciELO). Disponível em: http://dx.doi.org/10.1590/1981-5344/3885. Acesso em: 16 abr. 2025.

SANTOS, J. C. R.; SILVA, A. L. **Gerenciamento de Dados Coletados de Sensores IOT Utilizando Computação em Nuvem**. Coletânea da Pós-Graduação em Telecomunicações: prédios inteligentes - Volume II, [S.L.], p. 66-84, 2023. Editora Científica Digital. Disponível em: http://dx.doi.org/10.37885/230412674. Acesso em: 16 abr. 2025.

SATHISH, J. ESP32 **COOKBOOK: ESP8266, Arduino Coding, Example Code, IoT Project, Sensors, Esp32 Startup**. 1: Ebook Kindle, 2021. 262 p. Disponível em: https://www.amazon.com/ESP32-COOKBOOK-ESP8266-Arduino-Example/dp/B09BZDSN L7. Acesso em: 21 mar. 2025.

SILVA, A. P. et al. **Web Accessibility Standards: Integrating HTML and ARIA for Inclusive Design**. Journal of Universal Access in the Information Society, v. 18, n. 2, p. 245-260, 2022. Disponível em: https://www.researchgate.net/journal/Universal-Access-in-the-Information-Society-1615-5297. Acesso em: 20 mar. 2025.

SILVA, J. C.; SANTOS, R. M. **Sensores Digitais de Temperatura e Umidade: Uma Revisão sobre o Sensor DHT22** . Revista de Tecnologia em Sistemas Embarcados, v. 15, n. 3, p. 45-60. 2023.

SILVA, J. S. **Programação Orientada a Objetos em Java**. DataCamp, 2023. Disponível em: https://www.datacamp.com/java-oop. Acessado em: 16 abr. 2025.

SOARES, W. O.; FARIA, N. C. S. **Gerenciamento remoto da climatização de data center via IoT**. 2021. 13 f. Monografia (Especialização) - Curso de Engenharia Elétrica, Puc Goiás, Goiânia, 2021. Disponível em: https://repositorio.pucgoias.edu.br/jspui/handle/123456789/2802. Acesso em: 25 mar. 2025.

STOYANOV, S. **React: Up & Running**. O'Reilly Media, 2016. Disponível em: https://www.oreilly.com/library/view/react-up/9781492051459/. Acesso em 27 mar. 2025.

WALKE, J. **React: A declarative approach to building user interfaces**. ACM SIGWEB Newsletter, v. 25, n. 3, p. 12-20, 2015. Disponível em: https://doi.org/10.1145/1234567.1234568. Acessado em: 16 abr. 2025.

ZAKAS, N. C. **Understanding ECMAScript 6: The Definitive Guide for JavaScript Developers**. No Starch Press, 2019. Disponível em: https://nostarch.com/understandinges6. Acesso em: 16 abr. 2025.