Universidade Federal de São Carlos– UFSCar
Centro de Ciências Exatas e de Tecnologia– CCET
Departamento de Computação– DC
Curso de Bacharelado em Engenharia da Computação– EnC

Rafael Tury Minatel

Desenvolvimento de aplicação para cidades inteligentes utilizando da plataforma FIWARE

São Carlos
2025

Rafael Tury Minatel

Desenvolvimento de aplicação para cidades inteligentes utilizando da plataforma FIWARE

Monografia apresentada ao Curso de Bacharelado em Engenharia da Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia da Computação.

Área de concentração: Engenharia de Software e Sistemas Inteligentes

Orientador: Fabiano Cutigi Ferrari

São Carlos
2025

Agradecimentos

Primeiramente, agradeço aos meus pais, Lucimara e Pedro Paulo, pelo apoio incondicional, pela compreensão nos momentos difíceis e por sempre acreditarem no meu potencial.

Ao meu orientador, Professor Dr. Fabiano Cutigi Ferrari, pela orientação dedicada, pela paciência e por todo o suporte ao longo desta jornada. Suas contribuições foram fundamentais para a realização deste trabalho.

A todos os professores do Departamento de Computação da UFSCar, pelos conhecimentos transmitidos e pela dedicação ao ensino, que contribuíram significativamente para minha formação acadêmica e profissional.

Ao Programa de Assistência Estudantil da UFSCar, pela fundamental ajuda financeira que possibilitou minha permanência na universidade e a conclusão deste curso.

A todos que, de alguma forma, contribuíram para minha formação acadêmica e para a realização deste trabalho, meu sincero muito obrigado.

"Audaciosamente ir onde ninguém jamais esteve."
(Star Trek: The Next Generation (1987))

Resumo

O crescimento acelerado das cidades contemporâneas demanda soluções tecnológicas que promovam eficiência na gestão de recursos e serviços urbanos. Nesse contexto, as Cidades Inteligentes utilizam Tecnologias da Informação e Comunicação para transformar dados em decisões eficazes. Este trabalho investiga e demonstra, através de uma abordagem prática, como a plataforma FIWARE pode ser aplicada no desenvolvimento de aplicações para Cidades Inteligentes. O objetivo principal foi desenvolver um protótipo funcional que simula o monitoramento e controle inteligente de um galpão de eventos, integrando sensores IoT (temperatura, umidade, CO_2, ocupação) e atuadores (climatizador, exaustor, alarme) através dos componentes FIWARE: Orion Context Broker, IoT Agent JSON e QuantumLeap. A metodologia adotou pesquisa aplicada com abordagem qualitativa e exploratória, desenvolvendo o sistema em ambiente containerizado com Docker e interface web em React. O sistema implementado demonstrou funcionamento adequado, com automação respondendo em menos de 2 segundos, visualização de dados históricos e interface responsiva. O trabalho documenta detalhadamente os desafios enfrentados (configuração de containers, CORS, integração entre componentes) e as soluções adotadas, apresenta análise comparativa com a plataforma InterSCity, e sistematiza requisitos e boas práticas para projetos similares. Os resultados demonstram que a FIWARE é uma plataforma viável, robusta e adequada para desenvolvimento de soluções de Cidades Inteligentes, desde que sejam compreendidas suas particularidades e adotadas práticas apropriadas de configuração e integração. As principais contribuições incluem um protótipo funcional replicável, documentação consolidada de integração entre componentes FIWARE, e análise crítica que complementa a literatura existente.

Palavras-chave: Cidades Inteligentes, FIWARE, Internet das Coisas, Orion Context Broker, Automação.

Abstract

The accelerated growth of contemporary cities demands technological solutions that promote efficiency in the management of urban resources and services. In this context, Smart Cities use Information and Communication Technologies to transform data into effective decisions. This work investigates and demonstrates, through a practical approach, how the FIWARE platform can be applied in the development of Smart City applications. The main objective was to develop a functional prototype that simulates intelligent monitoring and control of an event hall, integrating IoT sensors (temperature, humidity, CO_2, occupancy) and actuators (air conditioner, exhaust fan, alarm) through FIWARE components: Orion Context Broker, IoT Agent JSON, and QuantumLeap. The methodology adopted applied research with a qualitative and exploratory approach, developing the system in a containerized environment with Docker and a web interface in React. The implemented system demonstrated adequate operation, with automation responding in less than 2 seconds, historical data visualization, and responsive interface. The work thoroughly documents the challenges faced (container configuration, CORS, component integration) and the solutions adopted, presents a comparative analysis with the InterSCity platform, and systematizes requirements and best practices for similar projects. The results demonstrate that FIWARE is a viable, robust, and suitable platform for developing Smart City solutions, provided its particularities are understood and appropriate configuration and integration practices are adopted. The main contributions include a replicable functional prototype, consolidated documentation of integration between FIWARE components, and critical analysis that complements the existing literature.

Keywords: Smart Cities, FIWARE, Internet of Things, Orion Context Broker, Automation.

Lista de ilustrações

Sumário

Capítulo 1

Introdução

Neste capítulo apresentam-se o contexto e a motivação para o trabalho realizado (Seção 1.1), os objetivos pretendidos (Seção 1.2), a metodologia seguida para a realização do trabalho (Seção 1.3) e uma breve descrição da organização do restante do documento (Seção 1.4).

1.1 Contexto e Motivação

O crescimento acelerado e a complexidade das cidades contemporâneas geram desafios significativos na gestão de recursos, mobilidade, segurança, saúde e qualidade de vida. Nesse contexto, torna-se essencial adotar soluções tecnológicas que promovam maior eficiência, integração e inteligência na administração dos serviços urbanos. É nesse cenário que se insere o conceito de Cidades Inteligentes (*Smart Cities*), cuja proposta é utilizar Tecnologias da Informação e Comunicação (TIC) para transformar dados em decisões mais eficazes, beneficiando o ambiente urbano e seus cidadãos (IBM, 2023).

Entre as soluções propostas, destaca-se a FIWARE FOUNDATION (2025), plataforma de código aberto que oferece componentes padronizados para o desenvolvimento de aplicações para Cidades Inteligentes. Sua capacidade de integrar sensores, processar dados em tempo real e interoperar com outras tecnologias a torna atrativa para projetos escaláveis e aderentes a padrões internacionais.

A motivação deste trabalho surge da necessidade de compreender, na prática, como a FIWARE pode ser empregada no desenvolvimento de soluções para Cidades Inteligentes, além de abordar a problemática da falta de padronização destacada por (LAI et al., 2020). Segundo (JOSé; RODRIGUES, 2024), um dos principais desafios está na escolha de tecnologias simples e eficazes, que promovam inovação digital sem gerar dependência.

A FIWARE já foi aplicada com sucesso em diversos contextos, como laboratórios urbanos de IoT (IMPAGLIAZZO et al., 2024), aplicações de alto nível para IoT (PATEL;

CASSOU, 2015), ecossistemas digitais e gêmeos digitais (TORREPADULA et al., 2025), manufatura (MENEXIS et al., 2024), edifícios inteligentes (BLECHMANN et al., 2023) e irrigação agrícola (PUIG; DíAZ; SORIANO, 2022). Além disso, a FIWARE é reconhecida como uma das principais plataformas em IoT (FORTINO et al., 2021), com estudos específicos que analisam sua segurança (PERATA; BETARTE, 2024), reforçando sua confiabilidade para aplicações críticas.

1.2 Objetivos

O objetivo deste Trabalho de Conclusão de Curso é investigar e demonstrar, por meio de uma abordagem prática, como a plataforma FIWARE pode ser aplicada no desenvolvimento de aplicações para ambientes inteligentes voltadas ao contexto urbano (FIWARE FOUNDATION, 2025). A proposta contempla a construção de um protótipo funcional que simule um ambiente de Cidade Inteligente, integrando sensores (Internet das Coisas – IoT), mecanismos de coleta de dados e serviços urbanos. Essa integração será viabilizada por meio dos principais componentes da FIWARE, com destaque para o Orion Context Broker, além de outros componentes disponibilizados pela plataforma. A partir do desenvolvimento desse protótipo, busca-se realizar uma análise da arquitetura da FIWARE, compreendendo sua estrutura modular, suas possibilidades de integração, e sua capacidade de escalar e interoperar com outras tecnologias.

Além da exploração técnica dos recursos oferecidos pela FIWARE, este trabalho também se propõe a destacar as vantagens observadas e os desafios enfrentados durante o processo de desenvolvimento de aplicações para Cidades Inteligentes, para sistematizar conhecimentos que possam servir de referência para futuros projetos. A intenção é propor diretrizes, recomendações e boas práticas que contribuam para a adoção mais eficiente da plataforma em iniciativas voltadas ao desenvolvimento de Cidades Inteligentes.

Por fim, o trabalho também contempla a avaliação do protótipo desenvolvido, com a realização de testes e a análise de resultados obtidos a partir dos dados inseridos na plataforma. Essa etapa permitirá verificar na prática a viabilidade da solução proposta, bem como levantar considerações relevantes sobre o uso da FIWARE em contextos urbanos reais.

1.3 Metodologia

Este trabalho utilizou o método de pesquisa aplicada, com foco na criação de um protótipo funcional que simula um ambiente de Cidade Inteligente utilizando a plataforma FIWARE. A abordagem foi qualitativa e exploratória, buscando compreender o funcionamento da plataforma, suas possibilidades de integração e seus limites técnicos a partir de uma experimentação prática.

O desenvolvimento foi conduzido por meio de uma pesquisa tecnológica, visando aplicar conhecimentos da área de computação para construir uma solução inovadora baseada em componentes da FIWARE. O trabalho também se configura como um estudo de caso experimental, uma vez que o protótipo será implementado em um cenário simulado e utilizado como base para avaliação.

As etapas do trabalho seguiram a seguinte sequência: levantamento de requisitos do protótipo; estudo da plataforma FIWARE com base nesses requisitos; documentação técnica e elaboração de diagramas; desenvolvimento da aplicação e sua integração com os componentes da FIWARE; documentação do desenvolvimento; e, posteriormente, obtenção e análise dos resultados obtidos e dificuldades encontradas/superadas do desenvolvimento do sistema. Durante esse processo, foram observadas as dificuldades e capacidades da plataforma de lidar com dados contextuais, sua escalabilidade e viabilidade de uso em aplicações urbanas reais.

1.4 Organização do Documento

Este documento está organizado em cinco capítulos, estruturados de forma a conduzir o leitor desde os conceitos fundamentais até os resultados obtidos e as conclusões do trabalho.

O Capítulo 2 apresenta a fundamentação teórica, abordando os conceitos de Cidades Inteligentes e Internet das Coisas, a plataforma FIWARE com seus componentes e arquitetura, e trabalhos relacionados que aplicaram a plataforma em contextos similares.

O Capítulo 3 descreve o planejamento e desenvolvimento do sistema, incluindo o levantamento de requisitos, os cenários de aplicação definidos, a arquitetura implementada utilizando componentes FIWARE e React, e as funcionalidades desenvolvidas para monitoramento, controle e automação.

O Capítulo 4 apresenta os resultados obtidos através dos testes realizados em cenários de uso realistas, bem como uma análise crítica dos desafios encontrados durante o desenvolvimento, as soluções adotadas, as limitações identificadas e as lições aprendidas ao longo do projeto.

O Capítulo 5 retoma os objetivos estabelecidos e discute como foram alcançados, destaca as principais contribuições práticas e teóricas do trabalho, reconhece as limitações e propõe direções para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta os conceitos fundamentais necessários para compreensão do trabalho desenvolvido. Primeiramente, são abordados os conceitos de Cidades Inteligentes e Internet das Coisas (Seção 2.1), seguidos pela apresentação da plataforma FIWARE, sua arquitetura e componentes utilizados (Seção 2.2). Por fim, são apresentados trabalhos relacionados que aplicam FIWARE em contextos similares (Seção 2.3).

2.1 Cidades Inteligentes e Internet das Coisas

2.1.1 Conceitos de Cidades Inteligentes

O conceito de Cidades Inteligentes (*Smart Cities*) emergiu como resposta aos desafios crescentes enfrentados pelos centros urbanos modernos, incluindo superpopulação, escassez de recursos, poluição ambiental e necessidade de serviços públicos mais eficientes (IBM, 2023). Uma Cidade Inteligente pode ser definida como um ambiente urbano que utiliza Tecnologias da Informação e Comunicação (TIC) para coletar dados de diversas fontes e aplicar esses dados para gerenciar recursos e serviços de forma mais eficiente, melhorando a qualidade de vida dos cidadãos.

Segundo Lai et al. (2020), um dos principais desafios no desenvolvimento de Cidades Inteligentes é a falta de padronização entre diferentes soluções e plataformas, o que dificulta a interoperabilidade e escalabilidade dos sistemas. José e Rodrigues (2024) destacam que a escolha de tecnologias adequadas deve equilibrar simplicidade, eficácia e inovação digital, evitando dependência tecnológica (*vendor lock-in*).

As aplicações de Cidades Inteligentes abrangem diversos domínios:

- **Gestão de tráfego e mobilidade urbana**: Monitoramento de fluxo de veículos, otimização de semáforos, sistemas de estacionamento inteligente.

- **Eficiência energética**: Iluminação pública adaptativa, monitoramento de consumo, gestão de redes elétricas inteligentes (*smart grids*).
- **Gestão ambiental**: Monitoramento da qualidade do ar, gestão de resíduos, controle de poluição sonora.
- **Segurança pública**: Sistemas de vigilância integrados, detecção de emergências, coordenação de resposta a incidentes.
- **Gestão de edifícios e espaços públicos**: Controle de climatização, monitoramento de ocupação, automação de sistemas prediais.

2.1.2 Internet das Coisas (IoT)

A Internet das Coisas (*Internet of Things* - IoT) é um paradigma fundamental para viabilização de Cidades Inteligentes. IoT refere-se à rede de dispositivos físicos (*things*) equipados com sensores, atuadores, software e conectividade de rede, permitindo coletar e trocar dados sem intervenção humana direta.

No contexto de Cidades Inteligentes, dispositivos IoT desempenham papel crucial:

- **Sensores**: Coletam dados do ambiente físico (temperatura, umidade, presença, qualidade do ar, nível de iluminação, etc.). São responsáveis pela camada de percepção da realidade urbana.
- **Atuadores**: Executam ações no ambiente físico baseadas em comandos (ligar/-desligar iluminação, ajustar climatização, acionar alarmes, etc.). Permitem que o sistema influencie o ambiente.
- **Dispositivos híbridos**: Combinam capacidades de sensoriamento e atuação, frequentemente incluindo processamento local (*edge computing*).

A integração de dispositivos IoT em sistemas de Cidades Inteligentes apresenta desafios técnicos importantes:

1. **Heterogeneidade de protocolos**: Dispositivos IoT utilizam diversos protocolos de comunicação (HTTP, MQTT, CoAP, LoRaWAN, etc.), requerendo camadas de abstração para integração.
2. **Escalabilidade**: Sistemas urbanos podem envolver milhares ou milhões de dispositivos, exigindo arquiteturas capazes de processar grandes volumes de dados.
3. **Confiabilidade**: Aplicações críticas (segurança, saúde) requerem alta disponibilidade e tolerância a falhas.

4. **Segurança e privacidade**: Dados coletados frequentemente incluem informações sensíveis, exigindo mecanismos robustos de proteção.

5. **Interoperabilidade**: Necessidade de integração entre sistemas de diferentes fornecedores e tecnologias.

É nesse contexto de desafios que se inserem plataformas padronizadas como a FIWARE, que oferece componentes modulares e baseados em padrões abertos para facilitar o desenvolvimento de aplicações IoT em contextos urbanos.

2.2 Plataforma FIWARE

2.2.1 Visão Geral e História

FIWARE é uma plataforma de código aberto (*open source*) para desenvolvimento de aplicações inteligentes em diversos domínios, com foco especial em Cidades Inteligentes (FIWARE FOUNDATION, 2025). Iniciada em 2011 como um projeto de parceria público-privada da União Europeia, a FIWARE evoluiu para um ecossistema global mantido pela FIWARE Foundation, uma organização sem fins lucrativos sediada em Berlim, Alemanha.

A filosofia central da FIWARE é promover interoperabilidade, escalabilidade e sustentabilidade no desenvolvimento de soluções inteligentes através de:

- **Padrões abertos**: Baseada no padrão NGSI (*Next Generation Service Interface*), promovendo interoperabilidade entre diferentes sistemas e evitando dependência de fornecedores específicos.

- **Arquitetura modular**: Componentes reutilizáveis chamados *Generic Enablers* que podem ser combinados conforme necessário para diferentes aplicações.

- **Código aberto**: Todos os componentes são disponibilizados com licenças de código aberto, permitindo auditoria, customização e uso sem custos de licenciamento.

- **Comunidade ativa**: Ampla adoção em projetos na Europa, América Latina, Ásia e África, com documentação extensa e comunidade de desenvolvedores ativa.

Segundo Fortino et al. (2021), a FIWARE é reconhecida como uma das principais plataformas em IoT, sendo utilizada em diversos contextos como laboratórios urbanos (IMPAGLIAZZO et al., 2024), manufatura (MENEXIS et al., 2024), edifícios inteligentes (BLECHMANN et al., 2023) e agricultura (PUIG; DíAZ; SORIANO, 2022).

2.2.2 Arquitetura FIWARE e Generic Enablers

A arquitetura da FIWARE é organizada em camadas funcionais, cada uma composta por *Generic Enablers* (GEs) - componentes modulares que implementam funcionalidades específicas. As principais camadas incluem:

- **Core Context Management**: Componentes centrais para gerenciamento de informações contextuais, incluindo o Orion Context Broker.
- **Interface IoT, Robots and Third-Party Systems**: Componentes para integração com dispositivos IoT e sistemas externos, incluindo IoT Agents.
- **Context Processing, Analysis and Visualization**: Componentes para processamento, análise e visualização de dados contextuais.
- **Context Data/API Management, Publication and Monetization**: Componentes para publicação e gerenciamento de APIs e dados.

Os *Generic Enablers* comunicam-se entre si através de APIs padronizadas, permitindo substituição e composição flexível. A seguir, são detalhados os componentes utilizados neste trabalho.

2.2.3 Orion Context Broker e Padrão NGSI

O Orion Context Broker é o componente central da plataforma FIWARE, responsável pelo gerenciamento de informações contextuais através do padrão NGSI. O conceito de **contexto** refere-se a informações sobre o estado de entidades do mundo real em um determinado momento.

2.2.3.1 Modelo de Dados NGSI

O padrão NGSI define um modelo de dados baseado em três conceitos fundamentais: entidades, atributos e metadados. Estes conceitos formam a base de toda comunicação e armazenamento de informações contextuais na plataforma FIWARE.

Entidades são o conceito central do modelo NGSI. Uma **entidade** representa uma abstração digital de qualquer objeto, físico ou conceitual, do mundo real que seja relevante para o domínio da aplicação. No contexto deste trabalho, entidades são utilizadas para representar tanto dispositivos físicos (sensores de temperatura, climatizadores, exaustores) quanto conceitos abstratos (salas, eventos, configurações). Cada entidade é uma representação virtual mantida no Orion Context Broker que armazena o estado atual do objeto real que ela modela.

Estruturalmente, uma entidade é composta por:

- **Identificador (`id`)**: Uma string única que identifica inequivocamente a entidade no sistema. Por exemplo, `"sensor_temp_001"` ou `"climatizador_principal"`.
- **Tipo (`type`)**: Uma string que categoriza a entidade, agrupando entidades com características similares. Por exemplo, `"TemperatureSensor"`, `"Room"`, `"AirConditioner"`. O tipo facilita consultas e operações sobre grupos de entidades.
- **Atributos**: Conjunto de propriedades que descrevem o estado e características da entidade. São os atributos que efetivamente armazenam as informações contextuais.

Atributos são as propriedades que compõem uma entidade e representam informações específicas sobre ela. Por exemplo, uma entidade do tipo `TemperatureSensor` pode ter atributos como `temperature` (valor atual de temperatura), `unit` (unidade de medida), e `location` (localização do sensor). Cada atributo contém:

- `name`: Nome do atributo
- `value`: Valor atual do atributo
- `type`: Tipo de dado (Number, String, Boolean, DateTime, geo:json, etc.)
- `metadata`: Informações adicionais sobre o atributo (timestamp de atualização, unidade de medida, precisão, etc.)

A Figura 1 ilustra um exemplo de entidade representando um sensor de temperatura.

```
{
  "id": "sensor_temp_001",
  "type": "TemperatureSensor",
  "temperature": {
    "type": "Number",
    "value": 23.5,
    "metadata": {
      "timestamp": {"type": "DateTime", "value": "2025-01-15T10:30:00Z"},
      "unit": {"type": "Text", "value": "Celsius"}
    }
  },
  "location": {
    "type": "Text",
    "value": "Sala Principal"
  }
}
```

Figura 1 – Exemplo de entidade NGSI representando um sensor de temperatura

Este modelo de dados permite que aplicações consultem, criem e atualizem o estado de entidades de forma padronizada, independentemente do tipo de dispositivo ou sistema

subjacente. A abstração proporcionada pelas entidades é fundamental para a interoperabilidade e escalabilidade da plataforma FIWARE.

2.2.3.2 Operações do Orion Context Broker

O Orion expõe uma API RESTful que permite operações CRUD (*Create, Read, Update, Delete*) sobre entidades:

- **Criação de entidades**: Registro de novas entidades no sistema
- **Consulta de entidades**: Recuperação de informações contextuais, com suporte a filtros e paginação
- **Atualização de atributos**: Modificação de valores de atributos existentes
- **Remoção de entidades**: Exclusão de entidades do sistema

Além das operações básicas, o Orion suporta **assinaturas** (*subscriptions*), um mecanismo fundamental que permite notificação automática de sistemas externos. Ao criar uma assinatura, um sistema externo especifica quais entidades/atributos monitorar e um endpoint HTTP para receber notificações. Quando ocorrem atualização em atributos monitorados, o Orion envia automaticamente uma requisição HTTP POST para cada endpoint registrado com os dados atualizados.

O Orion permite múltiplas assinaturas para as mesmas entidades, processando notificações de forma paralela e independente. Em cenários com muitas assinaturas, o desempenho é gerenciado através de *thread pools* configuráveis. Para alto volume de notificações, é recomendado utilizar filas de mensagens intermediárias (RabbitMQ, Kafka) para evitar sobrecarga dos sistemas notificados.

2.2.4 IoT Agent JSON

Os *IoT Agents* são componentes FIWARE que atuam como tradutores entre protocolos IoT específicos e o padrão NGSI do Orion Context Broker. A FIWARE oferece IoT Agents para diversos protocolos (MQTT, UltraLight, LWM2M, LoRaWAN, OPC-UA, etc.).

O **IoT Agent JSON** foi utilizado neste trabalho por sua simplicidade e adequação ao contexto de desenvolvimento de protótipo. Suas principais funcionalidades incluem:

- **Registro de dispositivos**: Configuração de dispositivos IoT (sensores e atuadores) com suas respectivas entidades associadas no Orion.
- **Tradução de medidas**: Recebe dados de sensores em formato JSON através de HTTP e atualiza automaticamente os atributos correspondentes no Orion.

- **Execução de comandos**: Permite envio de comandos a atuadores através do Orion. O fluxo é: aplicação → Orion (atualiza atributo de comando) → IoT Agent (recebe notificação) → dispositivo físico.

- **Provisionamento em grupo**: Permite registrar múltiplos dispositivos com configurações similares através de *service groups*.

O IoT Agent JSON simplifica significativamente a integração de dispositivos IoT, abstraindo complexidades de protocolo e permitindo que desenvolvedores foquem na lógica de aplicação ao invés de detalhes de comunicação.

2.2.5 QuantumLeap e Persistência de Dados Históricos

Enquanto o Orion Context Broker mantém apenas o estado atual das entidades (última leitura de sensores), muitas aplicações requerem acesso a dados históricos para análise de tendências, geração de relatórios e aprendizado de máquina.

O **QuantumLeap** é o *Generic Enabler* oficial da FIWARE para persistência de séries temporais. Suas características incluem:

- **Integração via assinaturas**: O QuantumLeap utiliza o mecanismo de assinaturas do Orion descrito anteriormente. Ao ser configurado (manualmente ou via API), uma requisição HTTP POST é enviada ao endpoint `/v2/subscriptions` do Orion, criando uma assinatura que especifica: (1) quais entidades/atributos monitorar, e (2) a URL do QuantumLeap como endpoint de notificação. A partir desse momento, cada atualização de atributos monitorados resulta em notificação automática ao QuantumLeap, que persiste os dados em série temporal.

- **API de consulta temporal**: Oferece endpoints RESTful para consultar histórico de atributos, com suporte a:

  - Filtros temporais (intervalo de datas)
  - Agregações (média, mínimo, máximo, soma)
  - Agrupamento temporal (por minuto, hora, dia, etc.)
  - Limit e offset para paginação

- **Suporte a múltiplos bancos de dados**: O QuantumLeap suporta diferentes backends para armazenamento, incluindo CrateDB (recomendado), TimescaleDB e PostgreSQL.

2.2.6 CrateDB

O **CrateDB** é um banco de dados SQL distribuído otimizado para dados de séries temporais e análise em tempo real. É o banco de dados oficialmente recomendado pela FIWARE para uso com QuantumLeap devido às seguintes características:

- **Escalabilidade horizontal**: Arquitetura distribuída baseada em *sharding*, permitindo adicionar nós para aumentar capacidade.
- **Interface SQL padrão**: Facilita consultas e integrações, utilizando sintaxe SQL familiar.
- **Otimização para séries temporais**: Estruturas de dados otimizadas para ingestão massiva de dados com *timestamp* e consultas de agregação temporal.
- **Alta performance em agregações**: Consultas analíticas (médias, somatórios, agrupamentos) são executadas eficientemente mesmo sobre grandes volumes de dados.
- **Integração nativa**: Drivers e configurações pré-estabelecidos para QuantumLeap, simplificando deployment.

O CrateDB armazena os dados em formato colunar otimizado, tornando consultas analíticas significativamente mais rápidas comparado a bancos de dados relacionais tradicionais orientados a linhas.

2.2.7 MongoDB

O **MongoDB** é um banco de dados NoSQL orientado a documentos, utilizado pelo Orion Context Broker e IoT Agents para persistência de configurações e estado das entidades. A escolha do MongoDB pela FIWARE é justificada por:

- **Modelo de documentos JSON**: Alinha-se naturalmente com o modelo de dados NGSI, onde entidades são representadas como documentos JSON.
- **Esquema flexível**: Permite evolução do modelo de dados sem necessidade de migrações complexas, adequado para ambientes IoT heterogêneos.
- **Performance em operações de leitura/escrita**: Otimizado para workloads de alta concorrência típicas de sistemas IoT.
- **Índices poderosos**: Suporta índices em campos aninhados e arrays, facilitando consultas complexas no Orion.

É importante destacar que o MongoDB armazena apenas o estado atual, sendo complementado pelo CrateDB/QuantumLeap para dados históricos, formando uma arquitetura completa de persistência.

2.3 Trabalhos Relacionados

A plataforma FIWARE tem sido aplicada em diversos contextos práticos, demonstrando sua versatilidade e eficácia. Esta seção apresenta brevemente alguns trabalhos relacionados que aplicaram FIWARE em cenários similares ao proposto neste trabalho.

2.3.1 Laboratórios Urbanos de IoT

Impagliazzo et al. (2024) apresentam a implementação de um *testbed* urbano de IoT utilizando FIWARE para experimentação de soluções de Cidades Inteligentes. O trabalho descreve a implantação de infraestrutura composta por sensores ambientais, câmeras e dispositivos de monitoramento distribuídos em ambiente urbano real, todos integrados através do Orion Context Broker e IoT Agents.

Os autores destacam os desafios de deployment em ambientes abertos, incluindo conectividade intermitente, proteção física de dispositivos e sincronização de dados. A infraestrutura desenvolvida permite que pesquisadores testem algoritmos de otimização, sistemas de aprendizado de máquina e aplicações de análise em tempo real sem necessidade de desenvolver toda a pilha de IoT do zero. O trabalho demonstra a capacidade da plataforma de integrar dispositivos heterogêneos (sensores de diferentes fabricantes e protocolos) e fornecer infraestrutura escalável para experimentação de aplicações urbanas em condições reais.

2.3.2 Aplicações de Alto Nível para IoT

Patel e Cassou (2015) exploram o desenvolvimento de aplicações de alto nível para IoT utilizando FIWARE, destacando como a plataforma abstrai complexidades de protocolos e comunicação de baixo nível. Os autores apresentam estudo de caso onde desenvolvedores sem conhecimento profundo de protocolos MQTT, CoAP ou HTTP conseguem criar aplicações funcionais interagindo apenas com a API NGSI do Orion Context Broker.

O trabalho demonstra que a camada de abstração fornecida pelos IoT Agents e pelo modelo de dados NGSI reduz significativamente o tempo de desenvolvimento e a curva de aprendizado. Desenvolvedores podem consultar e atualizar estados de dispositivos através de simples requisições HTTP REST, sem necessidade de implementar clientes específicos para cada protocolo IoT. Esta abstração é particularmente valiosa em ambientes com dispositivos heterogêneos, onde a padronização via NGSI facilita integração e manutenção do sistema.

2.3.3 Ecossistemas Digitais e Gêmeos Digitais

Torrepadula et al. (2025) investigam o uso da FIWARE para construção de ecossistemas digitais e gêmeos digitais (*digital twins*) urbanos. Gêmeos digitais são réplicas virtuais de sistemas físicos que são continuamente atualizadas com dados em tempo real, permitindo monitoramento, análise e simulação do comportamento do sistema real.

Os autores demonstram como o modelo de entidades NGSI do Orion Context Broker é adequado para representar objetos urbanos (edifícios, ruas, veículos, infraestrutura) e suas propriedades dinâmicas. A integração com componentes de visualização 3D e ferramentas de simulação permite criar representações virtuais interativas do ambiente urbano. O trabalho explora casos de uso como simulação de cenários de tráfego, otimização de consumo energético e planejamento urbano baseado em dados históricos e em tempo real. A capacidade da FIWARE de manter sincronização contínua entre o mundo físico (através de sensores) e o modelo digital é identificada como fator chave para viabilidade de gêmeos digitais urbanos.

2.3.4 Manufatura Inteligente

Menexis et al. (2024) aplicam FIWARE no contexto de manufatura inteligente, demonstrando que a plataforma não se limita a aplicações urbanas.O trabalho descreve a implementação de sistema de monitoramento de linha de produção onde sensores industriais (temperatura de máquinas, vibração, consumo energético, contadores de produção) são integrados ao Orion Context Broker.

Os autores destacam requisitos específicos do ambiente industrial, como latência reduzida para controle em tempo real, confiabilidade elevada (disponibilidade 24/7) e integração com sistemas legados (PLCs, SCADA). A arquitetura proposta utiliza IoT Agents customizados para protocolos industriais (OPC-UA, Modbus) e implementa lógica de controle através de componentes FIWARE de processamento de eventos. O trabalho demonstra que a modularidade da FIWARE permite adaptação para diferentes domínios, mantendo benefícios de padronização e interoperabilidade mesmo em ambientes industriais com requisitos rigorosos de desempenho e confiabilidade.

2.3.5 Edifícios Inteligentes

Blechmann et al. (2023) apresentam aplicação da FIWARE para gestão de edifícios inteligentes, incluindo controle de climatização, iluminação e monitoramento de ocupação. Este trabalho é particularmente relevante por tratar de cenário próximo ao desenvolvido nesta pesquisa.

Os autores descrevem a implementação de sistema integrado para edifício comercial, utilizando sensores de temperatura, umidade, CO_2, presença e luminosidade distribuídos em múltiplas salas. O sistema implementa lógica de automação para otimização energé-

tica, ajustando climatização e iluminação com base em ocupação e condições ambientais. A arquitetura utiliza Orion Context Broker para centralização de dados, IoT Agent MQTT para integração de sensores, QuantumLeap para análise histórica de consumo, e interface web para visualização e controle manual.

Os resultados reportados incluem redução de 23% no consumo energético através de automação inteligente e melhoria na satisfação dos ocupantes devido ao controle mais preciso de condições ambientais. O trabalho destaca como a FIWARE facilita integração de diferentes subsistemas (HVAC, iluminação, segurança) através de modelo de dados unificado, evitando silos de informação típicos de soluções proprietárias.

2.3.6 Irrigação Agrícola Inteligente

Puig, Díaz e Soriano (2022) exploram o uso da FIWARE para sistemas de irrigação agrícola inteligente, demonstrando aplicação em contexto rural. O trabalho aborda desafios específicos do ambiente agrícola, como cobertura limitada de rede, alimentação por energia solar e exposição a condições ambientais extremas.

Os autores implementam sistema de monitoramento distribuído com sensores de umidade do solo, temperatura, precipitação e estações meteorológicas, integrados via protocolo LoRaWAN devido ao longo alcance necessário em áreas rurais. O sistema utiliza lógica de decisão baseada em limiares de umidade e previsão meteorológica para acionar automaticamente válvulas de irrigação, otimizando uso de água.

A integração de dados históricos através do QuantumLeap permite análise de padrões sazonais e ajuste de parâmetros de irrigação conforme características específicas de cada área cultivada. O trabalho demonstra como FIWARE pode ser adaptada para cenários com conectividade limitada e requisitos de baixo consumo energético, expandindo seu escopo além de ambientes urbanos densamente conectados.

2.3.7 Análise de Segurança

Perata e Betarte (2024) conduzem análise de segurança da plataforma FIWARE, avaliando vulnerabilidades e propondo melhores práticas para deployment seguro em ambientes de produção. Os autores utilizam metodologia de testes de penetração e análise estática de código nos principais Generic Enablers.

O trabalho identifica aspectos críticos de segurança, incluindo autenticação e autorização de APIs (recomendando uso de OAuth2 e JSON Web Tokens), proteção contra injeção de código em consultas NGSI, criptografia de comunicações (TLS/SSL obrigatório em produção), e isolamento de rede entre componentes. Os autores propõem arquitetura de referência com camadas de segurança, incluindo firewalls, API gateways, e segmentação de rede.

Adicionalmente, o trabalho aborda segurança em nível de dados, discutindo anonimização de informações sensíveis e conformidade com regulamentações de privacidade (GDPR, LGPD). As recomendações incluem auditoria regular de logs, atualização constante de componentes, e configuração adequada de permissões de acesso. Este trabalho reforça a viabilidade da FIWARE para aplicações críticas quando boas práticas de segurança são seguidas, fornecendo guia prático para implantações seguras.

2.3.8 Posicionamento deste Trabalho

Este trabalho diferencia-se dos relacionados ao focar especificamente no desenvolvimento de um sistema completo de monitoramento e controle para galpões/centros de eventos, integrando:

- ❏ Múltiplos sensores ambientais (temperatura, umidade, CO_2, ocupação)
- ❏ Atuadores inteligentes com acionamento manual e automático
- ❏ Sistema de alertas em tempo real para condições críticas
- ❏ Regras de automação configuráveis pelo usuário
- ❏ Interface web completa para visualização e controle
- ❏ Análise de dados históricos através de gráficos

Além disso, o trabalho contribui com uma análise detalhada dos desafios práticos enfrentados durante o desenvolvimento e soluções adotadas, fornecendo orientações para futuros projetos similares.

Capítulo 3

Planejamento e Desenvolvimento

Este capítulo descreve o planejamento e as decisões tomadas para o desenvolvimento do sistema proposto. Inicia-se com o levantamento de requisitos que guiaram a implementação (Seção 3.1), seguido pelos cenários de aplicação previstos (Seção 3.2), pela arquitetura e implementação (Seção 3.3), e, por fim, pelas funcionalidades desenvolvidas (Seção 3.4).

3.1 Levantamento de requisitos

O levantamento de requisitos do protótipo foi realizado considerando os objetivos definidos para o trabalho e as restrições técnicas e temporais do projeto. A Tabela 1 apresenta os requisitos funcionais e não-funcionais, bem como as restrições que delimitam o escopo da solução proposta.

Tabela 1 – Requisitos Funcionais, Não-Funcionais, e Restrições do Sistema

ID	Descrição	Categoria
RF01	O sistema deve permitir a simulação de sensores de temperatura, umidade, presença e CO_2 em um ambiente de galpão/centro de eventos.	Requisito Funcional
RF02	O sistema deve permitir a simulação de atuadores, como lâmpadas inteligentes, climatizadores e dispositivos de alarme.	Requisito Funcional
RF03	O sistema deve possibilitar o cadastro de sensores e atuadores (reais ou simulados).	Requisito Funcional

Continua na próxima página

ID	Descrição	Categoria
RF04	O sistema deve enviar os dados coletados dos sensores para componentes responsáveis por armazenar e manipular tais dados.	Requisito Funcional
RF05	O sistema deve manter o estado atualizado de cada sensor e atuador registrado.	Requisito Funcional
RF06	O sistema deve armazenar dados históricos de sensores usando	Requisito Funcional
RF07	O dashboard deve exibir em tempo real a temperatura, umidade, CO_2, número de pessoas no local e o estado dos atuadores.	Requisito Funcional
RF08	O dashboard deve apresentar gráficos e indicadores com dados históricos provenientes do banco de dados.	Requisito Funcional
RF09	O usuário deve poder acionar atuadores (ex.: ligar climatização, ativar exaustores, acionar alarmes) diretamente pelo dashboard.	Requisito Funcional
RF10	O sistema deve monitorar a quantidade de pessoas dentro do galpão e alertar caso o limite de capacidade seja ultrapassado.	Requisito Funcional
RF11	O sistema deve monitorar os níveis de CO_2 e acionar automaticamente o sistema anti-incêndio caso o valor ultrapasse o limite de segurança definido.	Requisito Funcional
RF12	O sistema deve gerar alertas visuais no dashboard e registros em log sempre que houver condições críticas (temperatura, CO_2, lotação máxima).	Requisito Funcional
RF13	O usuário deve poder configurar limiares (thresholds) para disparo automático de atuadores (por exemplo: temperatura acima de X $\rightarrow$ ligar climatizador).	Requisito Funcional
RNF01	O sistema deve ser executado em containers Docker utilizando Docker Compose.	Requisito Não Funcional
RNF02	A comunicação entre os componentes deve utilizar o padrão NGSI definido pela FIWARE.	Requisito Não Funcional
RNF03	O sistema deve permitir a adição de novos sensores e atuadores de forma escalável.	Requisito Não Funcional
RNF04	O sistema deve apresentar interface responsiva e de fácil usabilidade no dashboard.	Requisito Não Funcional
RNF05	O sistema deve possibilitar integração futura com mecanismos de autenticação/autorização de usuários.	Requisito Não Funcional

Continua na próxima página

ID	Descrição	Categoria
RNF06	O sistema deve registrar e manter logs de eventos críticos (por exemplo: alarmes, acionamento de atuadores automáticos).	Requisito Não Funcional
R01	O protótipo deve ser implementado utilizando exclusivamente componentes do ecossistema FIWARE (Orion, IoT Agent, QuantumLeap).	Requisito Não Funcional
R02	O sistema deve utilizar banco de dados para persistência do Orion e IoT Agent.	Requisito Não Funcional
R03	O sistema deve utilizar o banco de dados CrateDB para persistência de séries temporais.	Requisito Não Funcional
R04	O dashboard deve ser desenvolvido em React, conforme definido no escopo do projeto.	Requisito Não Funcional
R05	O protótipo deve ser executado em ambiente local, não sendo exigida publicação em nuvem.	Requisito Não Funcional

3.2 Cenários de Aplicação

Para validar o sistema desenvolvido e demonstrar suas capacidades, foram definidos cenários de aplicação que representam situações reais de uso em contextos de cidades inteligentes. Esta seção descreve os cenários planejados e como cada um exercita diferentes aspectos dos requisitos.

3.2.1 Cenário Principal: Monitoramento de Galpão/Centro de Eventos

O cenário principal, que guiou o levantamento de requisitos, simula o gerenciamento inteligente de um galpão ou centro de eventos. Este ambiente apresenta desafios interessantes, como, por exemplo:

- ❏ Variação significativa de ocupação (de vazio a lotado)
- ❏ Necessidade de controle ambiental (temperatura, umidade, qualidade do ar)
- ❏ Requisitos de segurança (limites de capacidade, detecção de incêndio)
- ❏ Oportunidades de automação (climatização, iluminação, ventilação)

3.2.1.1 Dispositivos do Cenário

O cenário contempla os seguintes dispositivos:

Sensores:

- **Sensor de temperatura e umidade**: Monitora continuamente as condições climáticas internas. É essencial para conforto térmico e pode indicar problemas no sistema de climatização.
- **Sensor de CO_2**: Mede a concentração de dióxido de carbono no ar, indicador de ventilação adequada e, em níveis muito elevados, potencial risco de incêndio ou acúmulo de gases.
- **Sensor de presença/contagem de pessoas**: Estima a quantidade de pessoas presentes no local. É fundamental para gestão de capacidade e segurança.

Atuadores:

- **Climatizador**: Sistema de ar condicionado que pode ser ligado ou desligado remotamente. É acionado automaticamente quando a temperatura ultrapassa limites configurados.
- **Exaustor/sistema de ventilação**: Pode ser ativado para renovação do ar quando o nível de CO_2 está elevado ou o ambiente está muito quente.
- **Sistema de alarme**: Dispositivo de alerta sonoro que pode ser acionado em situações críticas (superlotação, incêndio, emergências).

3.2.1.2 Fluxos de Uso

Fluxo 1 - Configuração inicial:

1. Administrador acessa a página de Serviços e cria um serviço IoT
2. Acessa a página de Dispositivos e registra todos os sensores e atuadores, associando-os ao serviço criado
3. Acessa a página de Assinaturas e cria uma *subscription* para que o QuantumLeap armazene dados históricos dos sensores
4. Acessa a página de Automação e configura regras:
   - Se temperatura $> 26°C$, ligar climatizador
   - Se $CO_2 > 700$ ppm, ligar exaustor
   - Se pessoas > 95, gerar alerta de capacidade próxima ao limite

Fluxo 2 - Operação normal:

1. Sensores enviam dados periodicamente (simulado através da interface)

2. Dashboard atualiza métricas em tempo real
3. Gráficos históricos mostram tendências ao longo do dia
4. Condições ambientais permanecem dentro dos limites normais

Fluxo 3 - Evento com público crescente:

1. Conforme evento começa, ocupação aumenta gradualmente
2. Temperatura e CO_2 começam a subir devido à presença de mais pessoas
3. Quando temperatura ultrapassa 26°C, regra automática aciona climatizador
4. Sistema gera alerta visual no dashboard: "Climatizador acionado automaticamente"
5. Operador visualiza o status do climatizador mudando para "Ativo" na página de Automação

Fluxo 4 - Condição crítica de CO_2:

1. Sensor detecta CO_2 ultrapassando 1000 ppm (limite crítico)
2. Sistema gera alerta de erro: "Nível Crítico de CO_2 - Sistema anti-incêndio deve ser verificado!"
3. Alerta é exibido com destaque na página de Automação
4. Se configurado na regra, exaustor é acionado automaticamente
5. Operador pode acionar manualmente outros sistemas através da página de Atuadores

Fluxo 5 - Superlotação:

1. Sensor de presença detecta ocupação $> 100\%$ da capacidade
2. Sistema gera alerta crítico: "Capacidade do Galpão: 110/100 pessoas (LIMITE ULTRAPASSADO!)"
3. Alerta aparece em destaque no topo da página de Automação
4. Operador pode acionar alarme manualmente para orientar evacuação parcial
5. Histórico de ocupação é registrado para análise posterior

Fluxo 6 - Análise pós-evento:

1. Administrador acessa Dashboard após o evento

2. Analisa gráficos históricos de temperatura, umidade, CO_2 e ocupação

3. Identifica horários de pico e correlação entre ocupação e condições ambientais

4. Pode ajustar regras de automação com base nos dados observados

5. Exporta dados para relatórios (funcionalidade futura)

3.2.2 Cenários Secundários

Embora o desenvolvimento tenha focado no cenário de galpão/evento, a arquitetura suporta adaptação para outros contextos:

1. Edifício Comercial Inteligente

   Sensores de ocupação por sala, controle de iluminação e climatização individualizada, monitoramento de consumo energético, otimização baseada em horários de expediente.

2. Espaço Público (Praça, Parque)

   Monitoramento de qualidade do ar, iluminação pública inteligente, contagem de visitantes, sistema de irrigação automatizado baseado em umidade do solo.

3. Estacionamento Inteligente

   Sensores de ocupação de vagas, sinalização digital, histórico de utilização, tarifação dinâmica baseada em demanda.

3.2.3 Mapeamento Cenários-Requisitos

A Tabela 2 apresenta como os cenários planejados exercitam os requisitos definidos. Este mapeamento foi utilizado como guia durante os testes, garantindo que todos os requisitos fossem validados através de cenários realistas de uso.

Tabela 2 – Mapeamento entre cenários e requisitos

Requisito	**Cenário que exercita**
RF01	Todos os fluxos (sensores)
RF02	Fluxos 3, 4, 5 (atuadores)
RF03	Fluxo 1 (configuração)
RF04	Fluxos 2-5 (envio de dados)
RF05	Todos os fluxos (estado no Orion)
RF06	Fluxo 6 (dados históricos)
RF07	Fluxos 2-5 (dashboard tempo real)
RF08	Fluxo 6 (gráficos históricos)
RF09	Fluxos 4, 5 (controle manual)
RF10	Fluxo 5 (monitoramento capacidade)
RF11	Fluxo 4 (monitoramento CO_2)
RF12	Fluxos 3, 4, 5 (alertas)
RF13	Fluxos 1, 3, 4 (regras automação)

3.3 Arquitetura e Implementação

A arquitetura do sistema foi projetada seguindo os princípios da plataforma FIWARE, utilizando containers Docker para facilitar o *deployment* e garantir a portabilidade da solução. Esta seção detalha as decisões de design, as tecnologias utilizadas e a estrutura dos componentes implementados.

3.3.1 Arquitetura Geral do Sistema

O sistema é composto por duas camadas principais: *backend* baseado em componentes FIWARE e *frontend* desenvolvido em React. A comunicação entre as camadas utiliza o padrão NGSI (*Next Generation Service Interface*), conforme especificado pela FIWARE Foundation, garantindo interoperabilidade e conformidade com os padrões de cidades inteligentes.

A Figura 2 ilustra a arquitetura geral do sistema, destacando os principais componentes e suas interações. A descrição dos componentes da arquitetura é apresentada a seguir.

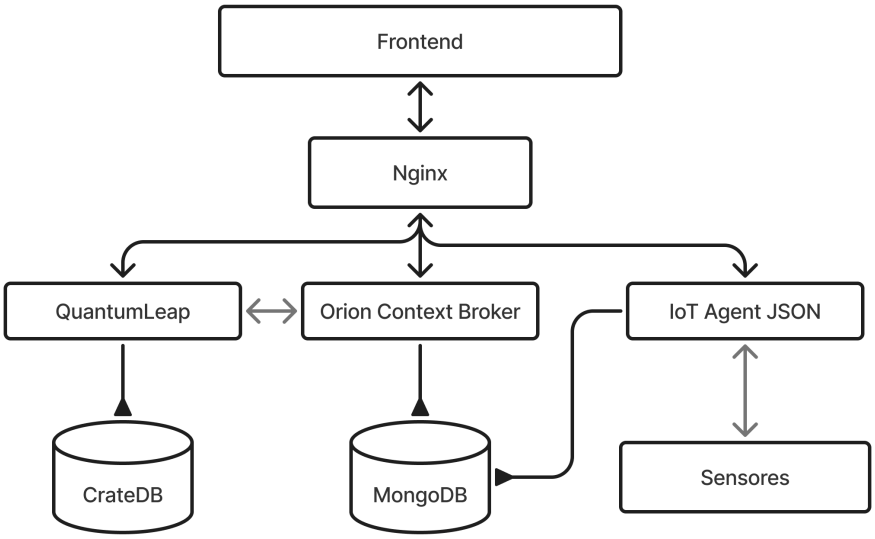

Figura 2 – Arquitetura geral do sistema desenvolvido.

3.3.2 Componentes FIWARE

A infraestrutura *backend* foi implementada utilizando Docker Compose, com os seguintes componentes FIWARE:

- **Orion Context Broker (versão 3.10.1)**: Responsável pelo gerenciamento das entidades contextuais seguindo o padrão NGSI. Exposto na porta 1026, é o componente central do sistema, mantendo o estado atualizado de todos os sensores e atuadores registrados (RF05).

- **IoT Agent JSON**: Atua como tradutor entre o protocolo JSON utilizado pelos dispositivos IoT e o padrão NGSI do Orion. Opera nas portas 4041 (API de provisionamento) e 7896 (endpoint para recebimento de dados), permitindo o registro e gerenciamento de dispositivos (RF03) e o envio de dados dos sensores para o Orion (RF04).

- **QuantumLeap (versão 1.0.0)**: Componente responsável por receber notificações do Orion Context Broker e persistir dados históricos em formato de séries temporais (RF06). Exposto na porta 8668, permite consultas de dados históricos para geração de gráficos e análises (RF08).

- **CrateDB (versão 5.2.3)**: Banco de dados distribuído otimizado para séries temporais, utilizado pelo QuantumLeap para armazenamento dos dados históricos. Exposto nas portas 4200 (interface HTTP) e 4300 (protocolo PostgreSQL).
- **MongoDB (versão 4.4)**: Banco de dados NoSQL utilizado pelo Orion Context Broker e IoT Agent para persistência de configurações e estado das entidades (R02). Exposto na porta 27017.
- **Nginx**: Servidor web configurado como *reverse proxy*, responsável por rotear requisições para os componentes internos e adicionar cabeçalhos CORS (*Cross-Origin Resource Sharing*) necessários para comunicação com o *frontend*.

Todos os componentes foram configurados em uma rede Docker isolada (`fiware`), garantindo comunicação segura entre os serviços e facilitando o gerenciamento da infraestrutura através do *script* `fiware-manager.sh`[1], que implementa comandos para iniciar (`start`), parar (`stop`), reiniciar (`restart`), monitorar status (`status`), visualizar logs (`logs`) e limpar o ambiente (`clean`) dos serviços (RNF01). O Código 3.1 ilustra a função de inicialização do ambiente.

Listing 3.1 – Função de inicialização do ambiente FIWARE

```
start_fiware() {
  echo "Iniciando ambiente FIWARE..."
  docker compose -f "$COMPOSE_FILE" up -d
  if [ $? -eq 0 ]; then
    echo "Todos os servicos iniciados com sucesso."
  else
    echo "Erro ao iniciar os servicos."
  fi
}
```

3.3.3 Frontend React

O *frontend* foi desenvolvido utilizando-se React 19 com TypeScript (R04), seguindo boas práticas de desenvolvimento moderno e componentização. A escolha dessas tecnologias se deu pela familiaridade e experiência do autor com as mesmas, permitindo maior produtividade no desenvolvimento e aproveitamento de conhecimentos prévios em projetos similares. As principais tecnologias utilizadas foram:

- React 19 + TypeScript
- Vite

[1] Código completo disponível em: <https://github.com/rafa-tm/aplicacao_fiware_tcc/blob/main/fiware/fiware-manager.sh>

- ❏ React Router 7
- ❏ Tailwind CSS 4
- ❏ Recharts
- ❏ shadcn/ui
- ❏ Lucide React

A estrutura de pastas do *frontend* foi organizada de forma modular, como ilustrado na Figura 3 e descrito abaixo:

- ❏ `src/pages/`: Componentes de página correspondentes às diferentes funcionalidades do sistema (Dashboard, Atuadores, Automação, Entidades, Dispositivos, Serviços, Assinaturas).
- ❏ `src/components/`: Componentes reutilizáveis, incluindo modais, formulários e componentes de visualização.
- ❏ `src/services/`: Módulo de integração com a API FIWARE, encapsulando as chamadas HTTP para os diferentes componentes.
- ❏ `src/hooks/`: Hooks React customizados para gerenciamento de estado e lógica de negócio.
- ❏ `src/routes/`: Configuração das rotas da aplicação.

3.3.4 Integração Frontend-Backend

A comunicação entre o *frontend* e os componentes FIWARE foi implementada através de uma camada de serviço (`services/api.ts`) que encapsula as requisições HTTP para os diferentes endpoints. Os componentes acessados pela camada de integração, descritos anteriormente na Subseção 3.3, são:

- ❏ Orion Context Broker (porta 1026): CRUD de entidades, consulta de tipos, gerenciamento de assinaturas.
- ❏ IoT Agent JSON (porta 4041): Registro de dispositivos, listagem de serviços, envio de comandos a atuadores.
- ❏ QuantumLeap (porta 8668): Consulta de dados históricos para atributos específicos de entidades.

As requisições utilizam o padrão REST, com formato JSON, e incluem, quando necessário, cabeçalhos FIWARE específicos como, por exemplo, `fiware-service` e `fiware-servicepath`, permitindo segmentação lógica de dispositivos e entidades.

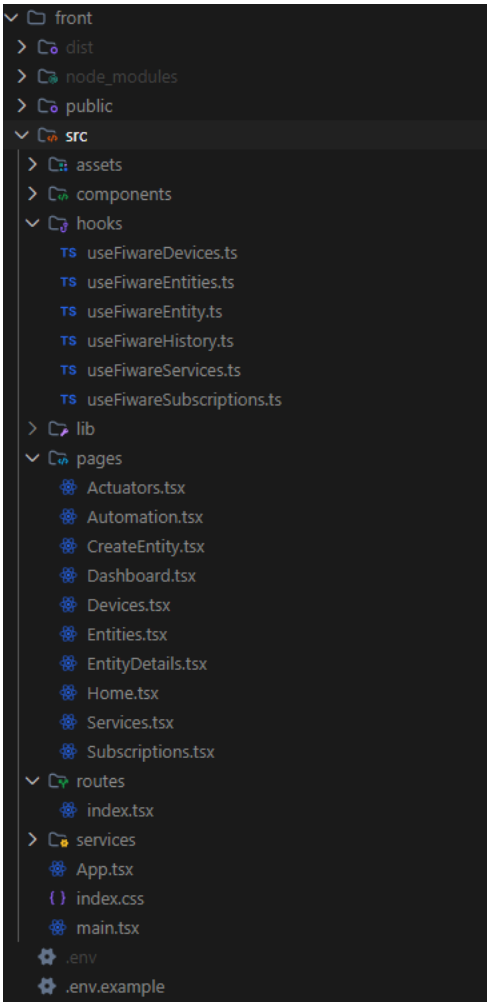

Figura 3 – Organização modular do projeto *frontend*.

3.4 Funcionalidades Desenvolvidas

Esta seção apresenta detalhadamente as funcionalidades implementadas no sistema, organizadas conforme as principais telas de interface de usuário da aplicação e seus respectivos propósitos.

3.4.1 Dashboard de Monitoramento

O dashboard do sistema (`Dashboard.tsx`), ilustrado na Figura 4, oferece uma visão consolidada de todas as entidades registradas no Orion Context Broker, atendendo aos requisitos RF07 e RF08. As principais funcionalidades incluem:

- **Cards de Resumo**: Exibem estatísticas gerais do sistema:
  - Total de entidades registradas
  - Número de tipos únicos de entidades
  - Total de atributos monitorados

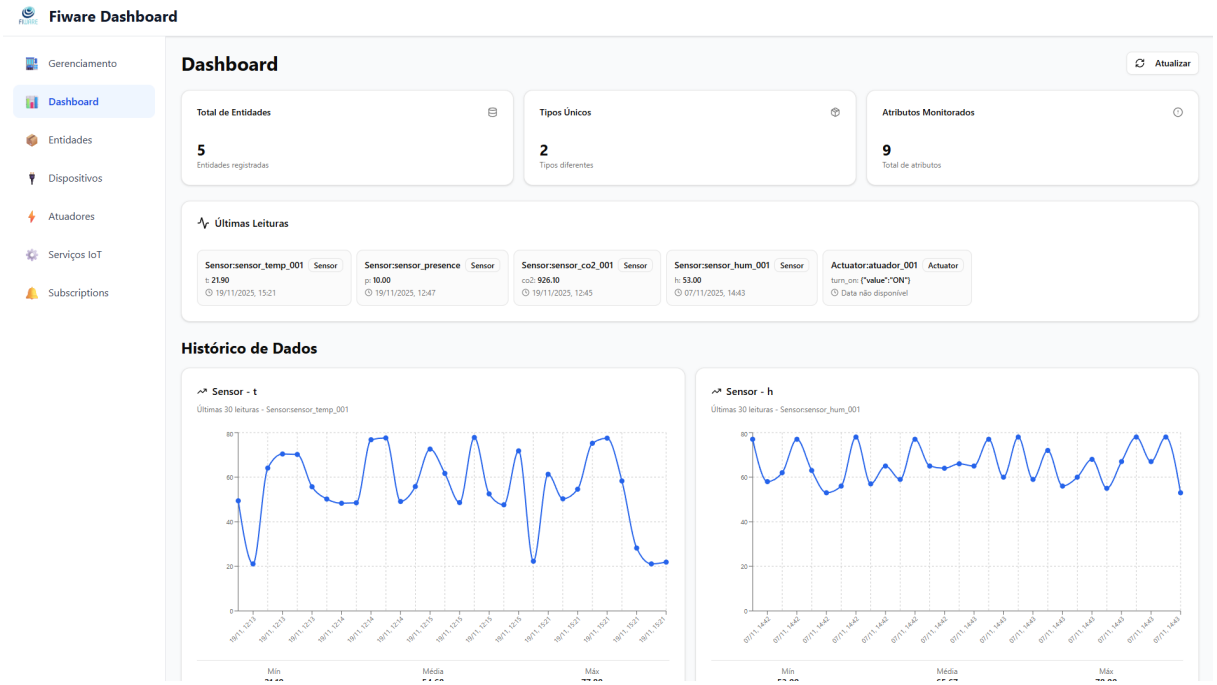

Figura 4 – Dashboard de monitoramento desenvolvida.

- ❏ **Card de Últimas Leituras**: Apresenta os valores mais recentes dos atributos de todas as entidades, permitindo visualização rápida do estado atual do sistema.

- ❏ **Gráficos de Histórico**: Para cada atributo numérico de cada entidade, é gerado um gráfico temporal utilizando dados do QuantumLeap. Os gráficos são organizados em grade responsiva e permitem análise visual de tendências e padrões nos dados coletados.

- ❏ **Tabela de Entidades**: Lista todas as entidades com informações de ID, tipo, número de atributos e ação para visualizar detalhes. Cada linha possui um link para a página de detalhes da entidade.

- ❏ **Atualização Manual**: Botão para forçar atualização dos dados a qualquer momento.

3.4.2 Gerenciamento de Entidades

A página de entidades (`Entities.tsx`), ilustrado na Figura 5, implementa operações CRUD completas para entidades no Orion Context Broker:

- ❏ **Listagem**: Exibe todas as entidades com filtros por tipo.

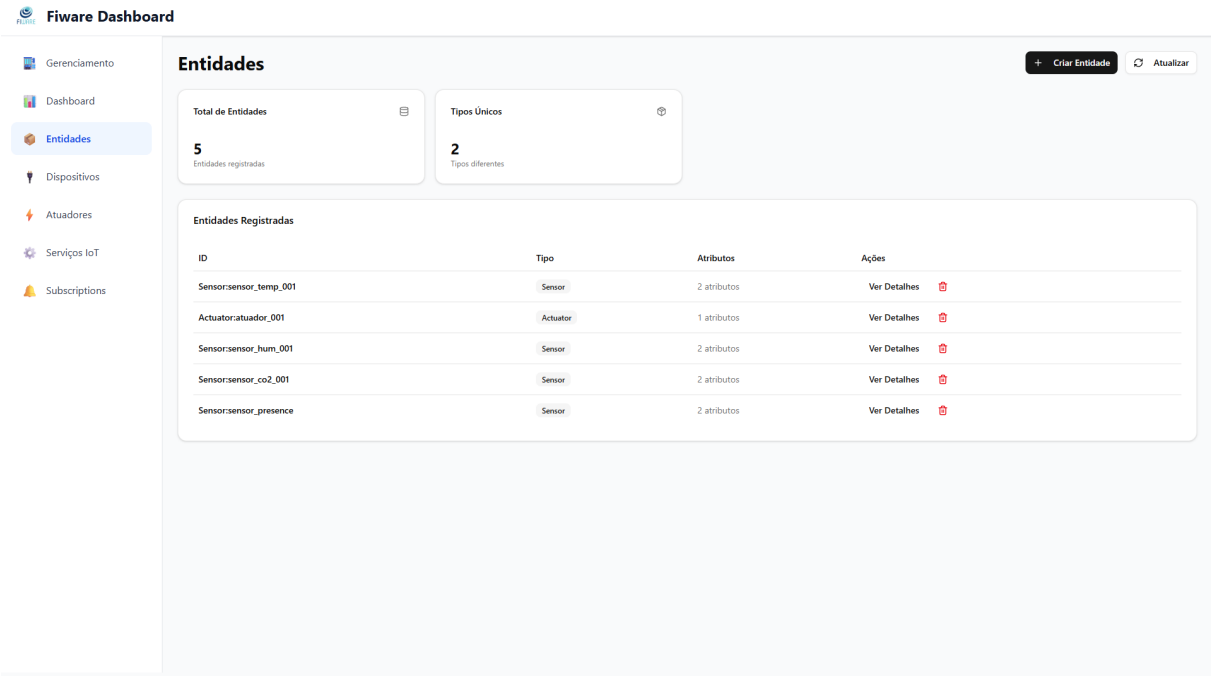

Figura 5 – Interface para gerenciamento de entidades.

- **Criação**: Modal[2] para criar novas entidades definindo ID, tipo e atributos.
- **Visualização**: Página de detalhes (`EntityDetails.tsx`) mostrando todos os atributos de uma entidade específica, com histórico visual através de gráficos.
- **Edição**: Modal para modificar valores de atributos existentes.
- **Exclusão**: Remoção de entidades com confirmação.

3.4.3 Gerenciamento de Dispositivos IoT

A página de dispositivos (`Devices.tsx`), Figura 6, implementa o registro e gerenciamento de dispositivos através do IoT Agent JSON, atendendo aos requisitos RF01, RF02 e RF03:

- **Registro de Dispositivos**: Modal (`RegisterDeviceModal.tsx`) que permite configurar:
  - ID do dispositivo
  - ID e tipo da entidade associada
  - Atributos do tipo `sensor` (para leitura de dados)

[2] Modal (ou janela modal) é um elemento de interface que aparece sobreposto à página principal, exigindo interação do usuário antes de retornar ao conteúdo subjacente. É comumente utilizado para formulários, confirmações e diálogos que requerem atenção imediata.

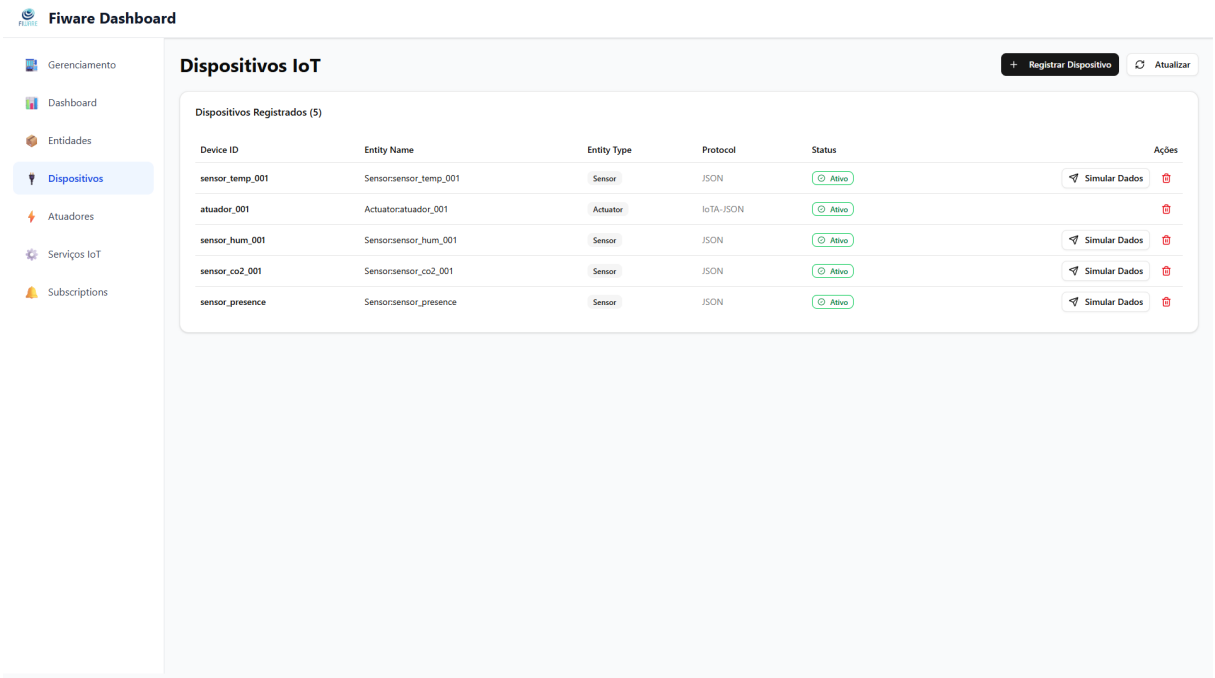

Figura 6 – Interface para gerenciamento de dispositivos IoT.

  - Atributos do tipo `command` (para controle de atuadores)
  - Configurações de transporte (protocolo, endpoint)

- ❏ **Listagem de Dispositivos**: Tabela com todos os dispositivos registrados, mostrando suas configurações e entidades associadas.

- ❏ **Envio de Dados de Sensores**: Modal (`SendSensorDataModal.tsx`), ilustrado na Figura 7, que permite simular o envio de dados de sensores através do endpoint do IoT Agent (RF04), demonstrando o fluxo completo de dados desde o dispositivo até o Orion Context Broker.

- ❏ **Exclusão de Dispositivos**: Remoção de dispositivos do IoT Agent.

3.4.4 Controle de Atuadores

A página de atuadores (`Actuators.tsx`), ilustrado na Figura 8, implementa o requisito RF09, oferecendo controle manual de dispositivos atuadores:

- ❏ **Listagem de Atuadores**: Filtra e exibe apenas entidades que possuem atributos do tipo `command`, ou seja, atuadores controláveis.

- ❏ **Estatísticas**: Cards mostrando o número total de atuadores, o número total de comandos disponíveis, e tipos diferentes de atuadores.

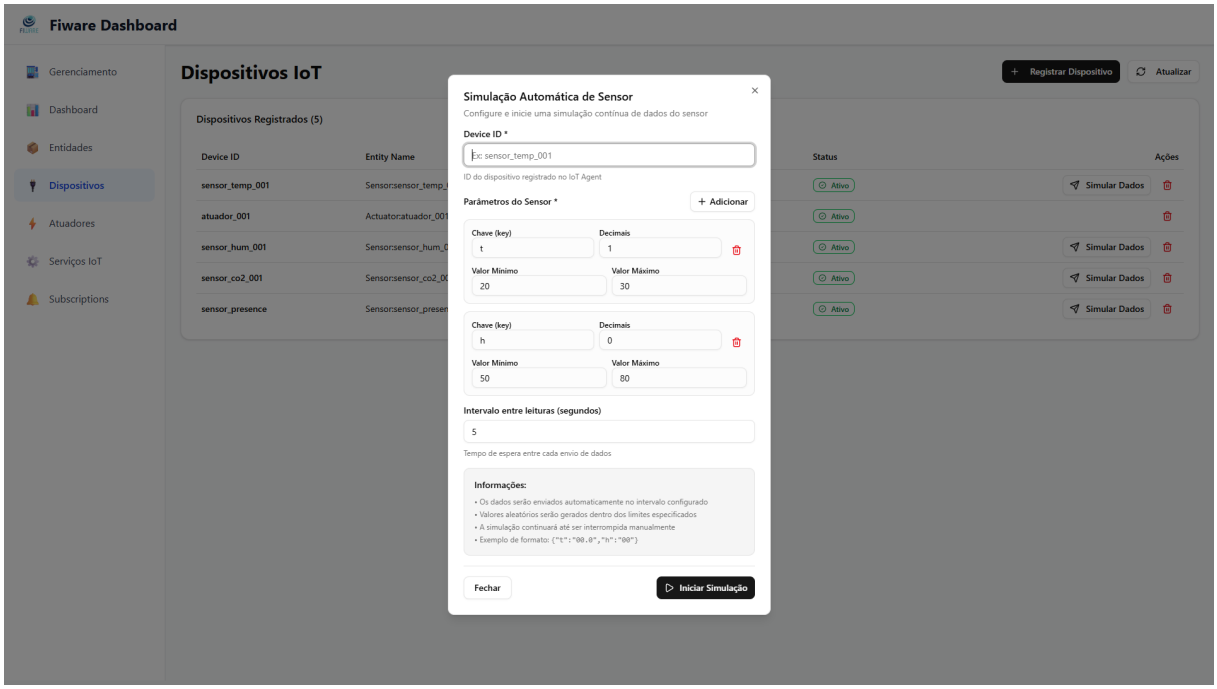

Figura 7 – Modal de envio de dados de sensores.

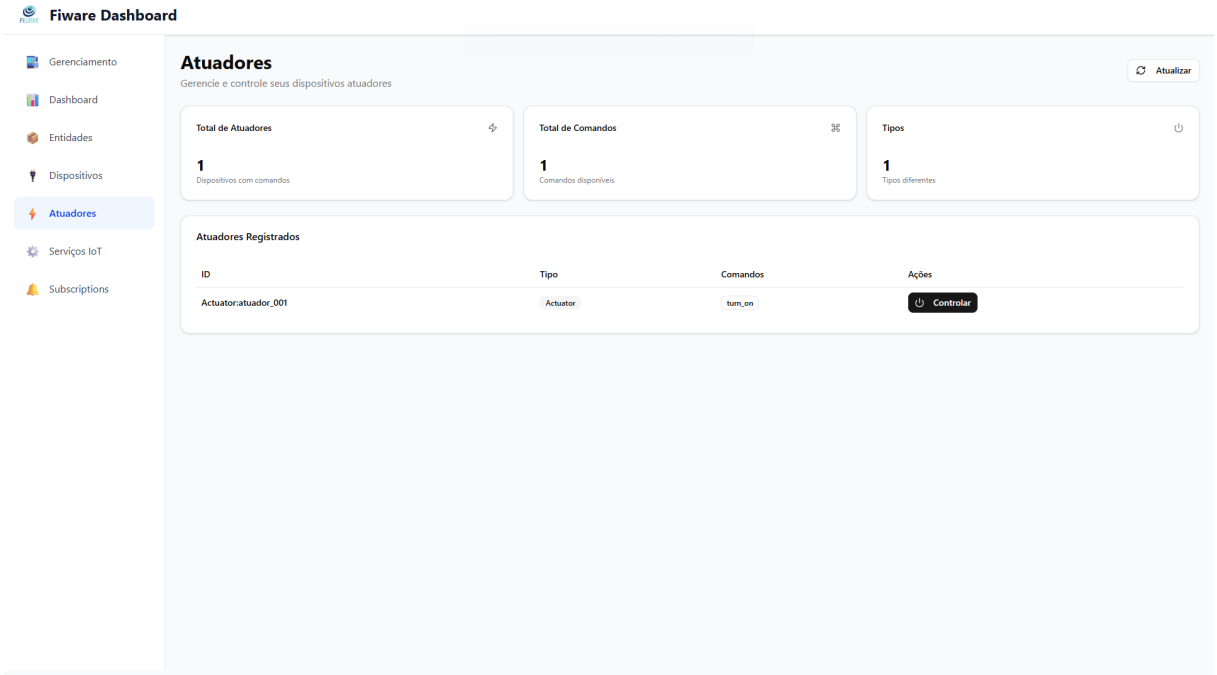

Figura 8 – Interface para visualização dos atuadores presentes no sistema.

❑ **Modal de Controle** (`ActuatorControlModal.tsx`): Interface para acionamento de comandos, permitindo:

– Seleção do comando a ser executado

- Definição do valor a ser enviado (texto livre ou JSON)
- Envio do comando através do IoT Agent
- Visualização da resposta do comando

❑ **Visualização de Status**: Exibe o estado atual de cada atuador baseado nos atributos `_status` e `_info` criados pelo Orion após execução de comandos.

3.4.5 Sistema de Automação e Monitoramento

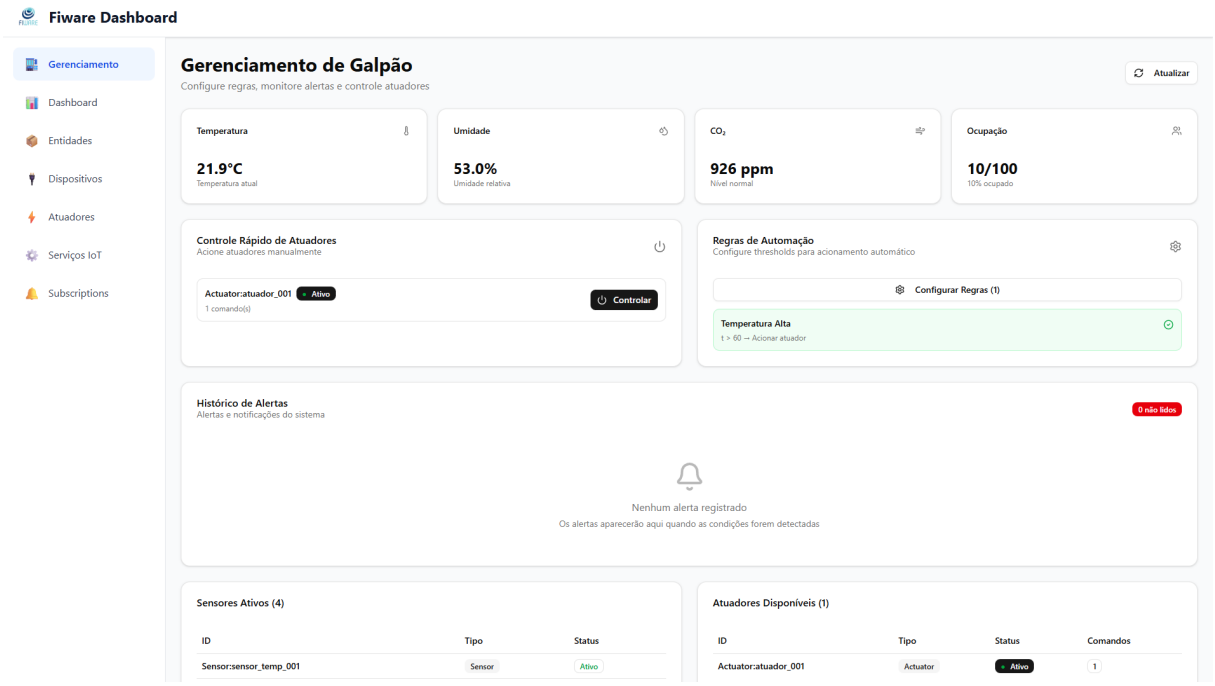

Figura 9 – Sistema de automação e monitoramento desenvolvido.

A página de automação (`Automation.tsx`), como pode ser visto na Figura 9, representa a funcionalidade central do sistema para gerenciamento de galpões/centros de eventos, implementando os requisitos RF10, RF11, RF12 e RF13. Esta página integra todas as capacidades do sistema em uma interface unificada de monitoramento e controle.

3.4.5.1 Monitoramento em Tempo Real

O sistema coleta e apresenta continuamente as seguintes métricas ambientais e operacionais:

❑ **Temperatura**: Exibida em graus Celsius (°C), monitorada através de sensores registrados no sistema.

- **Umidade**: Apresentada em percentual (%), indicando a umidade relativa do ar no ambiente.
- **CO_2**: Medida em partes por milhão (ppm), com destaque visual quando ultrapassa o limite de segurança de 1000 ppm.
- **Ocupação**: Contagem de pessoas presentes no local versus capacidade máxima definida (RF10), com indicação percentual e alertas visuais quando próximo ou acima do limite.

Essas métricas são apresentadas em cards na parte superior da página, com ícones representativos e cores indicativas do estado (normal, atenção, crítico). O sistema realiza atualização automática dos dados a cada segundo através de *polling*, garantindo que as informações exibidas estejam sempre sincronizadas com o estado real do ambiente.

3.4.5.2 Sistema de Alertas Automáticos

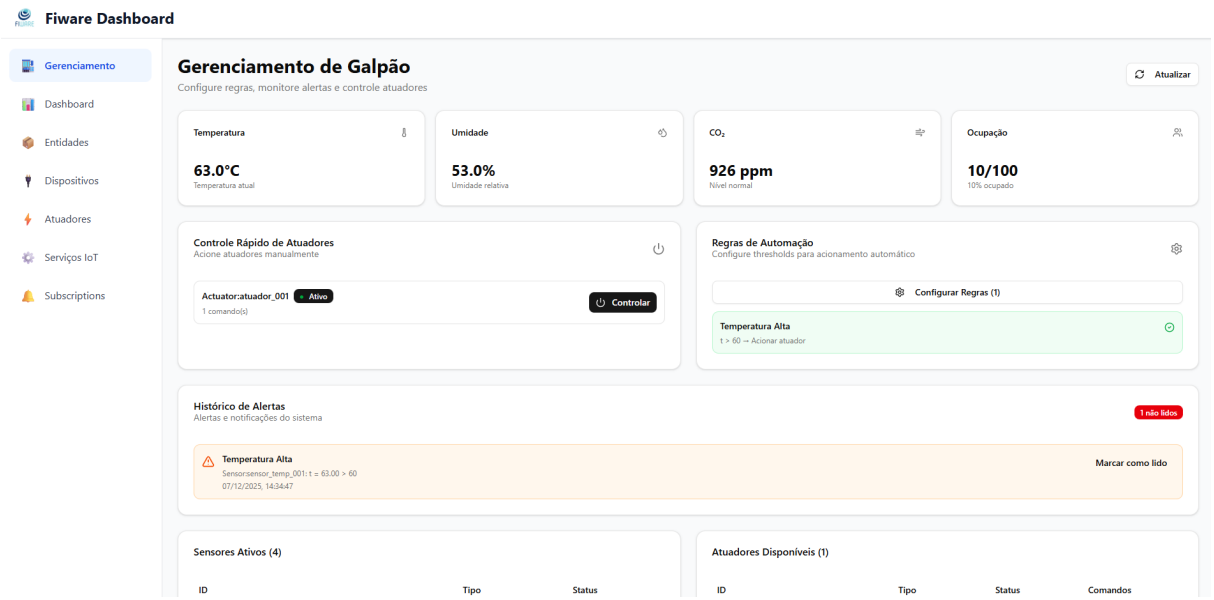

Figura 10 – Interface de gerenciamento para galpões com alertas ativos

O sistema implementa um mecanismo robusto de geração e gerenciamento de alertas (RF12) (Fig.10), classificados em três níveis de severidade:

- **Info**: Alertas informativos sobre eventos do sistema.
- **Warning**: Alertas de atenção quando métricas se aproximam de limites críticos.
- **Error**: Alertas críticos quando limites de segurança são ultrapassados.

Os alertas são gerados automaticamente nas seguintes situações:

1. **Capacidade do Galpão** (RF10):
   - ❏ Alerta de *warning* quando ocupação > 90% da capacidade
   - ❏ Alerta de *error* quando ocupação > 100% da capacidade
2. **Nível de CO_2** (RF11):
   - ❏ Alerta de *error* quando CO_2 > 1000 ppm
   - ❏ Mensagem específica indicando necessidade de verificação do sistema anti-incêndio
3. **Regras Personalizadas**:
   - ❏ Alertas gerados quando condições definidas pelo usuário são atendidas
   - ❏ Baseados em thresholds configuráveis (RF13)

Para evitar saturação do sistema e spam de notificações, foi implementado um mecanismo de *cooldown*[3] de 10 segundos entre alertas do mesmo tipo. Cada alerta inclui:

- ❏ Título descritivo
- ❏ Mensagem detalhada com valores específicos
- ❏ Timestamp de ocorrência
- ❏ Indicador de leitura (lido/não lido)
- ❏ Ícone e cor correspondentes ao nível de severidade

Os alertas são exibidos em um card de histórico, com destaque para alertas críticos não lidos na parte superior da página. O sistema permite marcar alertas como lidos individualmente e limpar alertas já lidos.

3.4.5.3 Regras de Automação

O sistema permite a configuração de regras personalizadas para acionamento automático de atuadores (RF13), através de um modal dedicado (`ThresholdRulesModal.tsx`), ilustrado na Figura 11. Cada regra possui os seguintes componentes:

- ❏ **Nome**: Identificação da regra.
- ❏ **Tipo de Entidade**: Define quais entidades serão monitoradas.
- ❏ **Atributo**: Especifica qual atributo será verificado (ex.: temperatura, CO_2, ocupação).

[3] Cooldown é um período de tempo de espera forçado entre execuções consecutivas de uma mesma ação, utilizado para prevenir sobrecarga do sistema e garantir que eventos sejam processados de forma controlada.

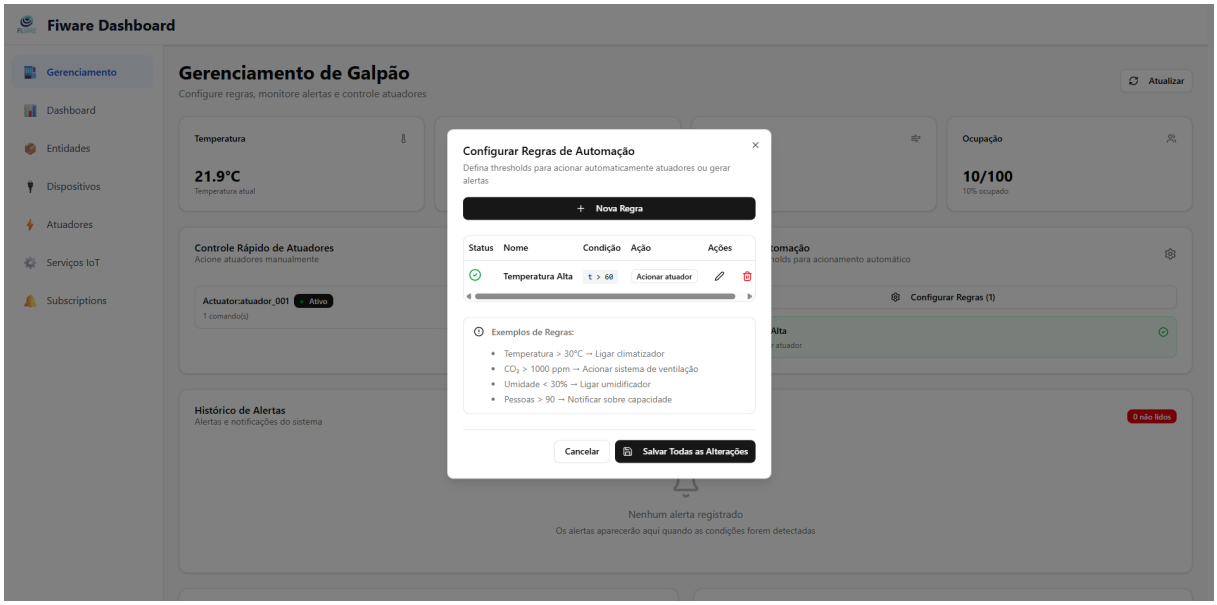

Figura 11 – Modal de gerenciamento de regras.

- ❏ **Operador**: Define o tipo de comparação (>, <, >=, <=, ==).
- ❏ **Valor Limite**: *Threshold* que dispara a regra.
- ❏ **Ação**: Define o que fazer quando a condição for atendida:
  - Gerar alerta apenas
  - Acionar atuador específico
- ❏ **Atuador e Comando**: Quando a ação é acionar atuador, especifica qual dispositivo e qual comando executar.
- ❏ **Valor do Comando**: Valor a ser enviado ao atuador (pode ser texto simples ou JSON).
- ❏ **Estado**: Habilitada ou desabilitada.

Exemplo de regra: “Se temperatura > 25°C, então acionar climatizador com comando ‘turn_on‘ e valor ‘ON‘”.

As regras são avaliadas continuamente (a cada segundo) sobre todos os sensores correspondentes. Quando uma condição é atendida:

1. Um alerta é gerado e adicionado ao histórico
2. Se configurado, o comando é enviado automaticamente ao atuador através da API do IoT Agent
3. O timestamp do disparo é registrado para controle de *cooldown*

4. Em caso de erro no envio do comando, um alerta adicional é gerado

As regras são persistidas no `localStorage` do navegador, mantendo a configuração mesmo após recarregar a página.

3.4.5.4 Controle Rápido de Atuadores

Além do acionamento automático, a página de automação oferece uma seção de controle rápido que lista os principais atuadores do sistema com suas informações de status em tempo real. Para cada atuador, são exibidos:

- ID do atuador
- Status atual (Ativo, Inativo, Pendente, Erro, Desconhecido)
- Número de comandos disponíveis
- Botão de controle para abertura do modal de acionamento manual

O status do atuador é determinado analisando os atributos `_info` e `_status` da entidade, que são atualizados pelo Orion após cada execução de comando. O sistema mapeia valores comuns (ON, OFF, ACTIVE, INACTIVE, etc.) para estados visuais consistentes com cores e ícones apropriados.

3.4.5.5 Visualização de Sensores e Atuadores Ativos

Na parte inferior da página, duas tabelas apresentam:

- **Sensores Ativos**: Lista todos os sensores detectados no sistema com seus tipos e status operacional.
- **Atuadores Disponíveis**: Lista todos os atuadores registrados com tipos, status e número de comandos.

Estas tabelas fornecem uma visão rápida da infraestrutura IoT disponível e seu estado de funcionamento.

3.4.6 Gerenciamento de Serviços e Assinaturas

O sistema também implementa páginas para configuração avançada:

- **Serviços** (`Services.tsx`): Permite criar e gerenciar serviços no IoT Agent, definindo configurações de grupo para dispositivos (RNF03), ilustrado na Figura 12.

❑ **Assinaturas** (`Subscriptions.tsx`): Implementa o gerenciamento de *subscriptions* do Orion Context Broker, permitindo que sistemas externos sejam notificados de mudanças em entidades específicas. Esta funcionalidade é essencial para integração do QuantumLeap, que se inscreve nas entidades para receber e persistir seus dados históricos (RF06), interface ilustrada na Figura 13.

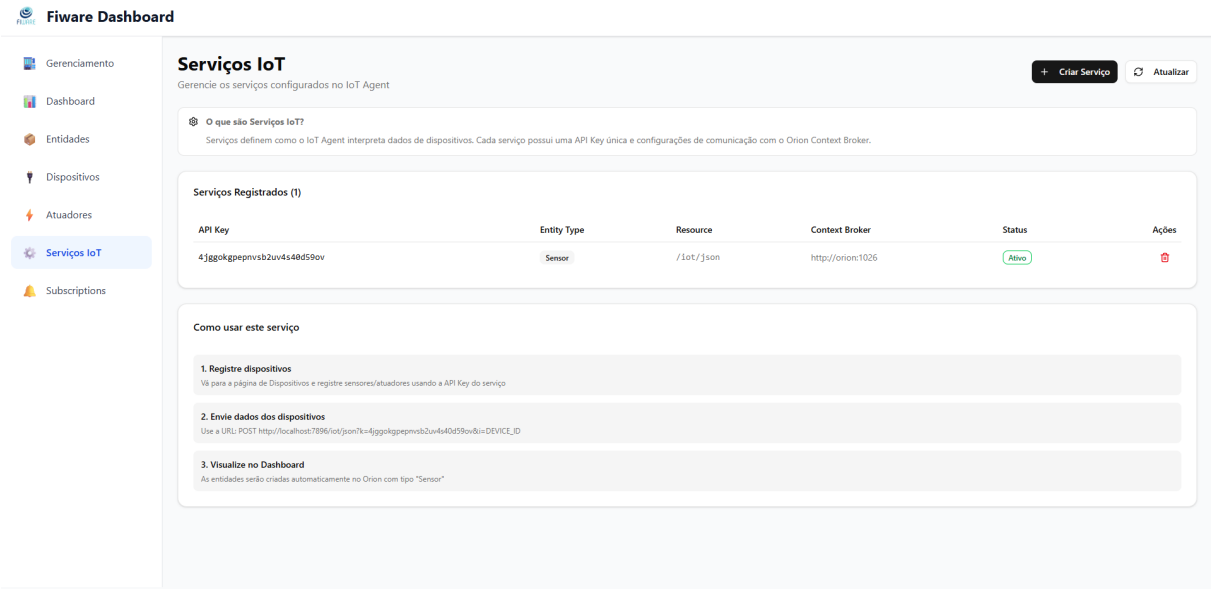

Figura 12 – Interface para gerenciamento de serviços.

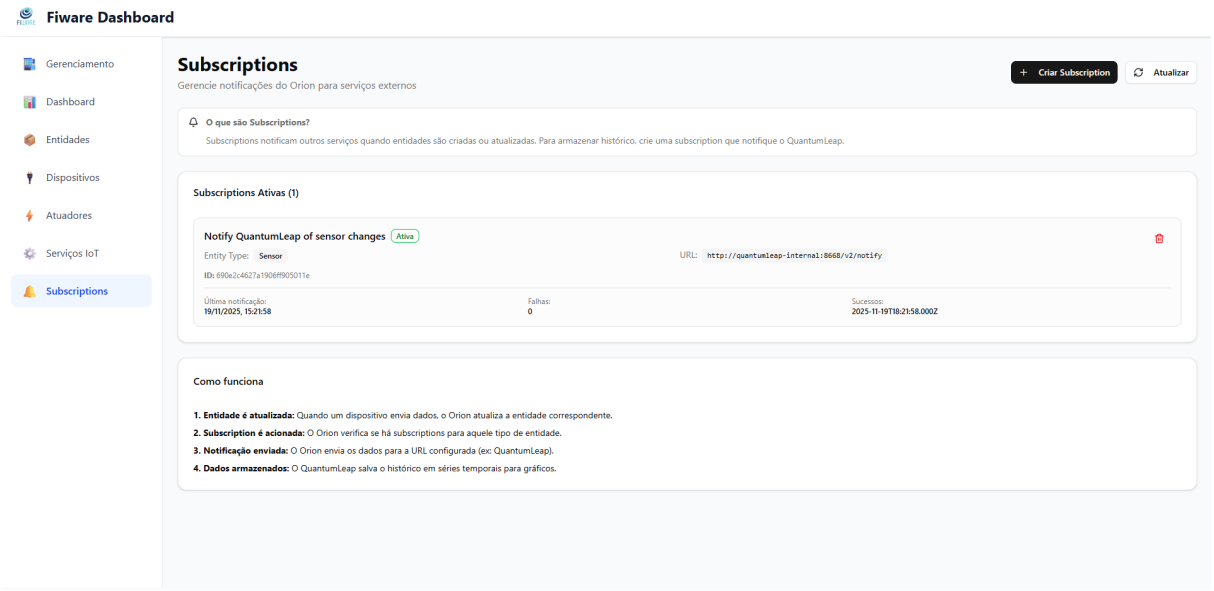

Figura 13 – Interface para gerenciamento de assinaturas.

3.5 Considerações Finais

Este capítulo apresentou o planejamento e desenvolvimento do sistema de gerenciamento inteligente baseado na plataforma FIWARE. Iniciou-se com o levantamento de requisitos funcionais e não-funcionais (Seção 3.1), que estabeleceram as diretrizes para a implementação do protótipo. Em seguida, foram definidos os cenários de aplicação (Seção 3.2), com foco principal no monitoramento de galpões e centros de eventos, demonstrando como o sistema pode ser aplicado em contextos reais de cidades inteligentes.

A arquitetura do sistema (Seção 3.3) foi detalhada, evidenciando a integração entre os componentes FIWARE (*backend*) e a interface React (*frontend*), todos orquestrados através de containers Docker. Por fim, as funcionalidades desenvolvidas (Seção 3.4) foram descritas de forma abrangente, incluindo dashboard de monitoramento, gerenciamento de entidades e dispositivos IoT, controle de atuadores e, principalmente, o sistema de automação com regras configuráveis e alertas em tempo real.

O código-fonte completo da implementação, incluindo configurações Docker, *scripts* de gerenciamento, código do *frontend* React e documentação técnica, está disponível publicamente em um repositório GitHub[4], permitindo reprodutibilidade e facilitando futuras extensões do trabalho.

O próximo capítulo apresentará os resultados obtidos com a implementação do sistema, incluindo testes de validação dos requisitos, análise dos cenários de uso e discussão sobre as contribuições e limitações do protótipo desenvolvido.

[4] Repositório do projeto: <https://github.com/rafa-tm/aplicacao_fiware_tcc>

Capítulo 4

Resultados, Análise e Discussão

Neste capítulo, são apresentados os resultados obtidos através dos testes realizados e uma análise crítica sobre os desafios encontrados durante o desenvolvimento. O capítulo está organizado em duas seções principais: resultados obtidos (Seção 4.1) e análise e discussão (Seção 4.2).

4.1 Resultados Obtidos

Esta seção apresenta os resultados práticos obtidos através da implementação e testes do sistema desenvolvido. Os testes foram realizados em ambiente local (R05), seguindo cenários de uso realistas para validação das funcionalidades implementadas.

4.1.1 Ambiente de Testes

O sistema foi testado na seguinte configuração:

- **Sistema Operacional**: Linux Ubuntu 22.04 LTS
- **Docker**: Versão 24.0.5
- **Docker Compose**: Versão 2.20.2
- **Node.js**: Versão 20.9.0
- **Navegador**: Google Chrome 120.0
- **Resolução de Tela**: 1920x1080 (testes de responsividade também realizados em 768x1024 e 375x667)

Todos os componentes FIWARE foram inicializados com sucesso através do script `fiware-manager.sh`, sem erros de inicialização ou dependências faltantes.

4.1.2 Cenário de Teste: Monitoramento de Evento

Para validação completa do sistema, foi criado um cenário simulando o monitoramento de um evento em um galpão. O cenário incluiu:

4.1.2.1 Configuração Inicial

1. Registro de serviço IoT no IoT Agent com configurações padrão (API key, protocolo JSON, endpoint de callback).
2. Registro de 4 dispositivos sensores:
   - ❏ Sensor de temperatura e umidade (`sensor_temp_001`)
   - ❏ Sensor de CO_2 (`sensor_co2_001`)
   - ❏ Sensor de presença/contagem de pessoas (`sensor_people_001`)
3. Registro de 3 dispositivos atuadores:
   - ❏ Climatizador (`actuator_climate_001`) com comandos `turn_on` e `turn_off`
   - ❏ Exaustor (`actuator_exhaust_001`) com comandos `start` e `stop`
   - ❏ Sistema de alarme (`actuator_alarm_001`) com comandos `activate` e `deactivate`
4. Criação de assinatura para persistência histórica dos sensores no QuantumLeap.

4.1.2.2 Simulação de Dados

Através da funcionalidade de envio de dados de sensores, foram simulados os seguintes cenários ao longo de 30 minutos:

Cenário 1 - Condições Normais (0-10 minutos):

- ❏ Temperatura: 22°C
- ❏ Umidade: 60%
- ❏ CO_2: 400 ppm
- ❏ Pessoas: 30 de 100 (30% de ocupação)

Cenário 2 - Aumento Gradual (10-20 minutos):

- ❏ Temperatura: 22°C → 28°C
- ❏ Umidade: 60% → 75%
- ❏ CO_2: 400 ppm → 800 ppm
- ❏ Pessoas: 30 → 85 (85% de ocupação)

Cenário 3 - Condições Críticas (20-30 minutos):

- ❏ Temperatura: 28°C → 32°C
- ❏ Umidade: 75% → 80%
- ❏ CO_2: 800 ppm → 1200 ppm (acima do limite de 1000 ppm)
- ❏ Pessoas: 85 → 110 (110% de ocupação, acima da capacidade)

4.1.2.3 Regras de Automação Configuradas

Para o teste, foram configuradas as seguintes regras:

1. **Regra de Temperatura**: Se temperatura > 26°C, acionar climatizador (comando `turn_on`)
2. **Regra de CO_2**: Se CO_2 > 700 ppm, acionar exaustor (comando `start`)
3. **Regra de Capacidade**: Se pessoas > 95, gerar alerta

4.1.3 Resultados Observados

4.1.3.1 Funcionalidades do Dashboard

- ❏ Os cards de resumo exibiram corretamente as estatísticas do sistema: 7 entidades totais (3 sensores + 3 atuadores + 1 serviço), 3 tipos únicos, e contagem adequada de atributos.
- ❏ O card de últimas leituras atualizou em tempo real conforme novos dados foram enviados aos sensores, mostrando os valores mais recentes de cada atributo.
- ❏ Os gráficos históricos foram gerados automaticamente para todos os atributos numéricos dos sensores, permitindo visualização clara das tendências ao longo do tempo. A consulta ao QuantumLeap retornou os dados corretamente formatados.
- ❏ A tabela de entidades listou todas as entidades registradas com links funcionais para suas páginas de detalhes.

4.1.3.2 Sistema de Alertas

Durante a execução dos cenários, o sistema gerou os seguintes alertas:

1. **T = 12 min**: Alerta de *warning* - "Regra de Temperatura: sensor_temp_001: temperature = 26.5 > 26"

2. **T = 15 min**: Alerta de *warning* - "Regra de CO_2: sensor_co2_001: CO2 = 750 > 700"

3. **T = 21 min**: Alerta de *warning* - "Capacidade do Galpão: Ocupação: 95/100 pessoas (Próximo do limite)"

4. **T = 23 min**: Alerta de *error* - "Nível Crítico de CO_2: CO_2: 1050 ppm (Limite: 1000 ppm) - Sistema anti-incêndio deve ser verificado!"

5. **T = 25 min**: Alerta de *error* - "Capacidade do Galpão: Ocupação: 110/100 pessoas (LIMITE ULTRAPASSADO!)"

Todos os alertas foram exibidos corretamente na interface, com cores e ícones apropriados ao nível de severidade. O mecanismo de *cooldown* funcionou adequadamente, evitando duplicação de alertas.

4.1.3.3 Acionamento Automático de Atuadores

As regras de automação funcionaram conforme esperado:

- **T = 12 min**: Climatizador foi acionado automaticamente com comando `turn_on`. O status do atuador na interface mudou de "Inativo" para "Ativo". Logs do sistema confirmaram o envio bem-sucedido do comando através do IoT Agent.

- **T = 15 min**: Exaustor foi acionado automaticamente com comando `start`. A mudança de status foi refletida imediatamente na interface.

O tempo de resposta entre a detecção da condição e o acionamento do atuador foi inferior a 2 segundos em todos os casos, demonstrando eficiência do sistema de automação.

4.1.3.4 Controle Manual de Atuadores

O acionamento manual através do modal de controle foi testado para todos os atuadores:

- Comandos foram enviados com sucesso ao IoT Agent

- O Orion Context Broker atualizou os atributos `_status` e `_info` das entidades

- O *frontend* refletiu as mudanças de estado em menos de 1 segundo

- Comandos com valores JSON complexos foram processados corretamente

4.1.3.5 Visualização de Dados Históricos

Os gráficos de séries temporais apresentaram corretamente:

- ❏ Curva de crescimento da temperatura de 22°C a 32°C
- ❏ Variação da umidade de 60% a 80%
- ❏ Aumento exponencial do CO_2 de 400 ppm a 1200 ppm
- ❏ Evolução da ocupação de 30 a 110 pessoas

Os gráficos utilizaram escala temporal no eixo X e escala numérica adequada no eixo Y, com linhas suaves e cores distintas para diferentes atributos.

4.1.4 Desempenho do Sistema

As seguintes medidas de desempenho foram observadas durante os testes:

- ❏ **Latência de Atualização**: Menos de um segundo entre envio de dados e atualização na interface
- ❏ **Tempo de Resposta da API**: Média de 150ms para operações do Orion, 200ms para QuantumLeap
- ❏ **Uso de Recursos**:
  - Containers Docker: 1.5 GB de RAM total
  - Frontend: Carregamento inicial em 500ms
  - Navegador: 200 MB de RAM para a aplicação
- ❏ **Polling**: Atualização a cada um segundo sem impacto perceptível na performance
- ❏ **Escalabilidade Testada**: Sistema operou estável com sete entidades e 100+ eventos de dados.

4.1.5 Interface do Usuário

A interface demonstrou boa usabilidade (RNF04), destacando-se:

- ❏ layout responsivo, que funcionou adequadamente em diferentes resoluções;
- ❏ navegação intuitiva entre as páginas através do menu lateral;
- ❏ feedback visual claro para ações (loading, sucesso, erro);
- ❏ cores e ícones que facilitaram a identificação rápida de estados e severidade;

- modais, que forneceram contexto apropriado sem sobrecarga de informação; e
- gráficos legíveis e informativos.

4.2 Análise e Discussão

Esta seção apresenta uma análise crítica do processo de desenvolvimento, discutindo os principais desafios encontrados, as soluções adotadas, as limitações identificadas e as lições aprendidas ao longo do projeto.

4.2.1 Desafios Encontrados

4.2.1.1 Configuração da Infraestrutura FIWARE

O primeiro grande desafio foi a configuração adequada dos componentes FIWARE em containers Docker. Especificamente:

- **Comunicação entre containers**: Inicialmente, houve dificuldades na comunicação entre o IoT Agent e o Orion Context Broker devido a configuração incorreta de hostnames na rede Docker. A solução adotada foi criar uma rede dedicada (`fiware`) e utilizar os nomes dos serviços como hostnames nas variáveis de ambiente.
- **CORS**: O acesso aos endpoints FIWARE a partir do frontend resultou em erros de CORS, uma vez que os componentes não incluem cabeçalhos CORS por padrão. A solução adotada foi adicionar um container Nginx como *reverse proxy*, configurado para incluir os cabeçalhos necessários (`Access-Control-Allow-Origin`, `Access-Control-Allow-Methods`, etc.) em todas as respostas.
- **Persistência de dados**: A configuração inicial não incluía volumes Docker, resultando em perda de dados ao reiniciar os containers. Volumes foram adicionados posteriormente para MongoDB e CrateDB, garantindo persistência.

4.2.1.2 Integração IoT Agent - Orion

A integração entre IoT Agent e Orion apresentou as seguintes complexidades:

- **Formato de dados**: Foi necessário compreender exatamente o formato JSON esperado pelo IoT Agent para tradução adequada para NGSI. A documentação oficial foi consultada extensivamente.
- **Comandos de atuadores**: A implementação de comandos exigiu entendimento do fluxo bidirecional: frontend → Orion → IoT Agent → dispositivo. Especialmente desafiador foi entender como o Orion cria automaticamente atributos `_status` e `_info` após execução de comandos.

- **Assinaturas**: A criação de assinaturas para o QuantumLeap requereu compreensão profunda do mecanismo de notificações do Orion, incluindo formato das notificações e endpoint correto do QuantumLeap.

4.2.1.3 Dados Históricos com QuantumLeap

A integração com QuantumLeap para dados históricos apresentou os seguintes desafios:

- **Formato de consulta**: A API do QuantumLeap possui sintaxe específica para consultas de séries temporais, diferente da API do Orion. Foi necessário estudar a documentação e realizar testes para construir consultas corretas.

- **Sincronização**: Inicialmente, havia delay significativo (> 10 segundos) entre atualização no Orion e disponibilidade no QuantumLeap. A busca pela causa dessa demora revelou que isto é comportamento normal devido ao modelo de assinaturas e processamento assíncrono.

- **Agregações**: Para gráficos com muitos pontos, consultas sem agregação retornavam volumes grandes de dados. Foi necessário implementar agregações temporais (médias por minuto/hora) para otimizar performance.

4.2.1.4 Sistema de Automação

A implementação do sistema de automação apresentou os seguintes desafios de projeto:

- **Polling vs WebSockets vs Mensageria**: A escolha entre polling HTTP e WebSockets para atualização em tempo real foi considerada. Optou-se por polling devido à simplicidade de implementação e ausência de suporte nativo a WebSockets no Orion. Também foi levantada a possibilidade de utilização de mensageria (utilizando de RabbitMQ ou Kafka), porém também descartou-se por complexidade de implementação e prazo.

- **Gerenciamento de alertas**: Foi necessário implementar um mecanismo de *cool-down* para evitar spam de alertas repetidos. A solução utilizou `useRef` no React para manter timestamps por tipo de alerta sem causar re-renderizações desnecessárias.

- **Avaliação de regras**: A avaliação contínua de regras poderia causar problemas de performance. Otimizações incluíram uso de `useMemo` para filtrar sensores/atuadores e verificação de habilitação de regras antes de avaliação.

4.2.2 Soluções Adotadas

As principais soluções técnicas adotadas incluem:

- **Arquitetura de serviços**: Criação de módulo `services/api.ts` que encapsula toda comunicação com FIWARE, facilitando manutenção e reutilização.
- **Hooks customizados**: Desenvolvimento de hooks React (`useFiwareEntities`, etc.) para gerenciar estado e lógica de negócio, promovendo reutilização de código.
- **TypeScript**: Uso extensivo de tipos e interfaces, o que garantiu segurança em tempo de desenvolvimento e facilitou refatorações.
- **Componentização**: Separação clara entre componentes de apresentação e lógica, facilitando testes e manutenção.
- **LocalStorage**: Para persistência de configurações do lado do cliente (regras de automação), evitando necessidade de backend adicional.

4.2.3 Limitações Identificadas

Algumas limitações do sistema atual foram identificadas:

- **Autenticação**: O sistema não implementa autenticação ou autorização. Qualquer usuário com acesso à URL pode controlar os dispositivos. Em ambiente de produção, seria necessário implementar camada de segurança (RNF05 prevê integração futura).
- **Validação de dados**: Não há validação robusta de dados de sensores. Valores absurdos (temperatura = 1000°C) são aceitos sem questionamento. Implementação de validação de faixas de valores seria benéfica.
- **Persistência de regras**: As regras de automação são armazenadas apenas no `localStorage`, o que significa que são específicas do navegador e não compartilhadas entre usuários ou dispositivos.
- **Backend na arquitetura**: Na versão desenvolvida para este trabalho foi realizado uma comunicação direta entre frontend e componentes da plataforma FIWARE, sem a existencia de um backend. Para uma correta divisão de responsabilidades e persistência de regras, seria necessário a inclusão e desenvolvimento de um backend separado para a aplicação e toda a lógica relacionada ao cenário de gerenciamento de galpão/centro de eventos.
- **Escalabilidade de gráficos**: Com muitas entidades (>100), a página de dashboard pode ficar sobrecarregada de gráficos, impactando performance. Paginação ou virtualização seriam necessárias.

- ❑ **Notificações**: Alertas são exibidos apenas na interface web. Não há integração com sistemas de notificação externa (email, SMS, push notifications).
- ❑ **Histórico de comandos**: Não há registro persistente de comandos executados (manual ou automaticamente). Implementar auditoria seria importante para ambientes de produção.
- ❑ **Simulação de dispositivos**: O sistema depende de envio manual de dados através da interface. Não há simulador automático de dispositivos, o que limitou testes de longa duração.
- ❑ **Ambiente local**: A restrição R05 limita o acesso ao sistema. Implantação (*deployment*) em nuvem com domínio público seria necessário para uso real.

4.2.4 Lições Aprendidas e Comparação com Trabalho Anterior

O desenvolvimento deste projeto proporcionou aprendizados significativos. Adicionalmente, é relevante contextualizar os resultados obtidos em relação a um trabalho anterior de objetivo similar desenvolvido por Citron (2025), que explorou a plataforma InterSCity (Del Esposte et al., 2017) para desenvolvimento de aplicações de Cidades Inteligentes.

4.2.4.1 Lições Aprendidas no Presente Trabalho

- ❑ **Importância do planejamento**: O levantamento detalhado de requisitos no Capítulo 3 foi fundamental para manter o foco e evitar *scope creep*[1]. Sempre que surgiram ideias de funcionalidades adicionais, a tabela de requisitos serviu como referência para avaliar se a adição era necessária.
- ❑ **Documentação é essencial**: A complexidade da plataforma FIWARE exigiu consulta constante à documentação oficial. Componentes de código aberto bem documentados facilitam significativamente a adoção e reduzem curva de aprendizado.
- ❑ **Testes incrementais**: Testar cada funcionalidade isoladamente antes de integração completa economizou tempo de depuração. Um exemplo de teste isolado refere-se à validação do envio de comandos através de `curl` antes de implementar interface web.
- ❑ **Docker simplifica deployment**: A containerização de todos os componentes facilitou enormemente o gerenciamento da infraestrutura. Um único comando (`docker compose up`) inicializa todo o backend, tornando o projeto portável e reproduzível.

[1] Scope creep (ou aumento descontrolado de escopo) é um fenômeno em gestão de projetos onde funcionalidades e requisitos são adicionados progressivamente além do escopo original, sem avaliação adequada de impacto em tempo, recursos e objetivos, comprometendo a conclusão do projeto.

- ❑ **Padrões facilitam integração**: O uso do padrão NGSI permitiu que componentes FIWARE de diferentes fornecedores (Orion da Telefónica, QuantumLeap da Orchestracities) funcionassem juntos sem problemas de compatibilidade.
- ❑ **Frontend reativo melhora UX**: A atualização automática da interface através de polling (mesmo que simples) melhorou significativamente a experiência do usuário, dando sensação de monitoramento em tempo real.
- ❑ **Trade-offs são inevitáveis**: Diversas decisões envolveram trade-offs (polling vs WebSockets, localStorage vs backend, etc.). Aprender a avaliar prós e contras e fazer escolhas pragmáticas foi uma habilidade desenvolvida.

4.2.4.2 Comparação com Trabalho Anterior

O trabalho de Citron (2025) apresentou uma aplicação para monitoramento de temperatura em estações de metrô utilizando a plataforma InterSCity. Ao comparar os dois trabalhos, emergem insights relevantes sobre escolhas de plataformas e abordagens para desenvolvimento de aplicações em Cidades Inteligentes.

Maturidade e Estabilidade das Plataformas:

Citron (2025) relatou desafios significativos com a estabilidade da plataforma InterSCity, incluindo indisponibilidade frequente dos serviços e problemas com dependências de pacotes (MongoDB, VirtualBox, Ruby) que geravam quebras de funcionamento. Em contraste, a plataforma FIWARE demonstrou maior maturidade e estabilidade, com desafios concentrados principalmente em configuração inicial e integração entre componentes, mas não em instabilidade dos serviços.

Escopo e Complexidade das Aplicações:

Enquanto o trabalho de Citron focou primariamente em monitoramento de temperatura com sistema de alertas (7 categorias), o presente trabalho expandiu significativamente o escopo para incluir múltiplos tipos de sensores (temperatura, umidade, CO_2, presença), atuadores controláveis (climatizador, exaustor, alarme), e um sistema completo de automação baseado em regras configuráveis. Esta diferença de escopo reflete tanto as capacidades das plataformas quanto os objetivos específicos de cada projeto.

Arquitetura e Infraestrutura:

A arquitetura do presente trabalho é substancialmente mais complexa, utilizando múltiplos microserviços FIWARE (Orion Context Broker, IoT Agent JSON, QuantumLeap, CrateDB, MongoDB, Nginx) em comparação com a estrutura mais simples do trabalho de Citron (3 componentes principais: criação de sensores, geração de dados, e interface). Esta complexidade adicional, embora exigindo maior esforço de configuração inicial, proporcionou funcionalidades avançadas como persistência robusta de séries temporais e capacidades de automação.

Persistência e Gerenciamento de Dados:

Uma limitação crítica identificada por Citron foi a impossibilidade de deletar dados de recursos na InterSCity, com espaço de armazenamento limitado e sem opções de limpeza. O presente trabalho, utilizando QuantumLeap com TimescaleDB, ofereceu solução mais robusta para armazenamento e consulta de dados históricos, demonstrando a maturidade do ecossistema FIWARE para aplicações que demandam análise temporal.

Funcionalidades Implementadas:

O trabalho de Citron concentrou-se em visualização de dados e geração de alertas. O presente trabalho expandiu estas capacidades incluindo: controle bidirecional de atuadores, sistema de automação com regras configuráveis, múltiplas interfaces especializadas (Dashboard, Entidades, Dispositivos, Atuadores, Automação), e integração com dados históricos para análise temporal. Esta expansão de funcionalidades demonstra o potencial da plataforma FIWARE para sistemas mais complexos de gerenciamento urbano.

Desafios Técnicos Específicos:

Ambos os trabalhos enfrentaram desafios relacionados à complexidade inerente de plataformas para Cidades Inteligentes. No entanto, os desafios diferiram em natureza: Citron enfrentou principalmente problemas de instabilidade da plataforma, enquanto o presente trabalho lidou com desafios de configuração e integração de múltiplos componentes (CORS, comunicação entre containers, formato de dados IoT, sincronização entre serviços).

Aprendizados Convergentes:

Apesar das diferenças de plataforma e escopo, ambos os trabalhos convergiram em lições fundamentais: a importância do planejamento detalhado para evitar *scope creep*, a necessidade de documentação abrangente para plataformas complexas, o valor de testes incrementais, e a relevância da containerização (Docker) para facilitar *deployment* e reprodutibilidade. Estas lições transcendem escolhas tecnológicas específicas e representam boas práticas gerais para desenvolvimento de sistemas IoT.

Escolha de Plataforma:

A comparação sugere que a escolha entre InterSCity e FIWARE deve considerar o trade-off entre simplicidade e capacidades. A InterSCity pode ser adequada para prototipagem rápida e projetos de menor escopo, especialmente em contextos acadêmicos brasileiros. A FIWARE, por sua vez, demonstrou ser mais apropriada para sistemas de produção que demandam maior robustez, escalabilidade, e funcionalidades avançadas como automação e análise histórica complexa.

Estes aprendizados, combinados com as limitações identificadas em ambos os trabalhos, fornecem direcionamento valioso para aprimoramentos futuros e orientações para projetos similares que utilizem plataformas de código aberto em contextos de Cidades

Inteligentes. As considerações sobre possíveis extensões e melhorias do trabalho são apresentadas no Capítulo 5.

4.3 Considerações Finais

Este capítulo apresentou os resultados obtidos com a implementação do sistema de gerenciamento inteligente baseado em FIWARE e uma análise crítica do processo de desenvolvimento. A Seção 4.1 detalhou os testes em cenário realista de monitoramento de galpão, validando o funcionamento das funcionalidades implementadas e o atendimento aos requisitos estabelecidos no Capítulo 3, com destaque para o sistema de automação que respondeu em menos de 2 segundos.

A Seção 4.2 apresentou os desafios técnicos enfrentados, as soluções adotadas, as limitações identificadas, as lições aprendidas e uma comparação com trabalho anterior de Citron (2025) que utilizou a plataforma InterSCity. A comparação revelou que, enquanto a InterSCity é adequada para prototipagem acadêmica, a FIWARE demonstrou maior robustez para sistemas que demandam automação avançada e persistência confiável de dados históricos. Ambos os trabalhos convergiram em lições sobre planejamento, documentação, testes incrementais e containerização.

As limitações identificadas apontam caminhos para evolução do sistema, e as lições aprendidas fornecem direcionamento para projetos similares em contextos de Cidades Inteligentes. O próximo capítulo apresenta as conclusões do trabalho, sintetizando as contribuições alcançadas e propondo direcionamentos para trabalhos futuros.

Capítulo 5

Conclusão

Este capítulo apresenta as considerações finais do trabalho desenvolvido, com destaque para os objetivos alcançados e contribuições obtidas (Seção 5.1), as limitações identificadas (Seção 5.2) e os trabalhos futuros sugeridos (Seção 5.3).

5.1 Objetivos e Contribuições

O objetivo principal deste Trabalho de Conclusão de Curso foi investigar e demonstrar, através de uma abordagem prática, como a plataforma FIWARE pode ser aplicada no desenvolvimento de aplicações inteligentes voltadas ao contexto de Cidades Inteligentes. Este objetivo foi alcançado através da implementação de um protótipo funcional que simula o monitoramento e controle inteligente de um galpão de eventos.

Conforme estabelecido na Seção 1.2, os objetivos específicos, com as contribuições associadas, foram:

1. **Construir um protótipo funcional e replicável**: Foi desenvolvido um sistema completo integrando sensores IoT (temperatura, umidade, CO_2, ocupação) e atuadores (climatizador, exaustor, alarme), comunicando-se através dos componentes FIWARE (Orion Context Broker, IoT Agent JSON e QuantumLeap). O sistema pode servir como base para projetos similares, com código disponível em repositório GitHub[1] e configurações Docker que facilitam reprodução do ambiente.

2. **Realizar análise aprofundada da arquitetura FIWARE**: O trabalho explorou a estrutura modular da FIWARE, documentando detalhadamente o processo de integração entre componentes, incluindo configurações específicas, formato de requisições e estrutura de dados - conhecimento que não é facilmente encontrado de forma consolidada na documentação oficial (Capítulos 2 e 3).

[1] <https://github.com/rafaelturytec/tcc-fiware-smart-city>

3. **Avaliar escalabilidade e interoperabilidade**: Através dos testes realizados (Capítulo 4), verificou-se que a arquitetura FIWARE suporta adição incremental de dispositivos e integração com diferentes tecnologias. Os dados de performance apresentados (latência, uso de recursos) fornecem referências quantitativas para dimensionamento de projetos similares.

4. **Documentar desafios e soluções**: A Seção 4.2 apresenta discussão detalhada dos desafios encontrados (configuração de containers, CORS via Nginx, gestão de assinaturas QuantumLeap, implementação de regras de automação) e as soluções adotadas, oferecendo orientações práticas para desenvolvedores que enfrentarão problemas similares em projetos futuros.

5. **Propor diretrizes e boas práticas**: Foram sistematizadas recomendações práticas que podem servir de referência para desenvolvedores em iniciativas similares, incluindo análise crítica de vantagens, limitações e *trade-offs* da plataforma, complementando a literatura que frequentemente foca apenas aspectos positivos.

6. **Validar através de cenários de teste**: A validação foi realizada através de cenários de uso realistas (Seção 4.1), demonstrando que todos os requisitos funcionais e não funcionais foram atendidos. A sistematização de requisitos (Tabela 1) e o mapeamento de cenários (Tabela 2) demonstram abordagem metodológica que pode ser aplicada em outros contextos.

Adicionalmente, o sistema implementa automação configurável pelo usuário (RF13), permitindo adaptação a diferentes cenários sem modificação de código - contribuição significativa para flexibilidade operacional.

Todos os objetivos foram cumpridos, demonstrando que a FIWARE é uma plataforma viável, robusta e adequada para desenvolvimento de soluções de Cidades Inteligentes, desde que sejam compreendidas suas particularidades e adotadas práticas apropriadas de configuração e integração.

5.2 Limitações do Trabalho

Apesar dos resultados obtidos, este trabalho apresenta algumas limitações que devem ser reconhecidas. As limitações técnicas específicas da implementação (autenticação, validação de dados, persistência de regras, escalabilidade de gráficos, notificações, histórico de comandos) foram detalhadas na Seção 4.2. Aqui são destacadas limitações de escopo e metodológicas mais gerais:

5.2.1 Limitações de Escopo

- **Simulação de dispositivos**: O sistema opera com dados simulados enviados manualmente através da interface web, não havendo integração com sensores e atuadores físicos reais. Embora isto não invalide a arquitetura proposta, testes com hardware real poderiam revelar desafios adicionais de latência, confiabilidade de comunicação e consumo de energia.

- **Cenário único**: O desenvolvimento focou especificamente no contexto de galpões (ou centros) de eventos. Embora a arquitetura seja adaptável, a validação em outros domínios (tráfego urbano, gestão energética, etc.) está fora do escopo deste trabalho.

- **Ambiente local e de desenvolvimento**: Conforme restrição R05, o sistema foi desenvolvido e testado exclusivamente em ambiente local, sem implementação de mecanismos de autenticação ou segurança adequados para produção. Implantação em nuvem ou ambiente de produção demandaria considerações adicionais de configuração, segurança, escalabilidade e controle de acesso.

5.2.2 Limitações Metodológicas

- **Validação com usuários reais**: O sistema não foi testado com usuários finais (administradores de eventos, operadores de segurança, etc.), limitando a avaliação de usabilidade a critérios técnicos e observação do desenvolvedor.

- **Testes de carga limitados**: Embora testes preliminares com 7 entidades simuladas tenham sido realizados, não houve validação em escala de centenas ou milhares de dispositivos.

- **Análise de longo prazo**: Os testes foram realizados em períodos curtos (até 30 minutos), não havendo dados sobre comportamento do sistema em operação contínua por dias ou semanas.

Estas limitações não comprometem a validade das conclusões apresentadas, mas indicam áreas onde trabalhos futuros poderiam expandir e aprofundar a pesquisa.

5.3 Trabalhos Futuros

Com base nas limitações identificadas e nos aprendizados obtidos durante o desenvolvimento, diversas direções para trabalhos futuros são propostas:

5.3.1 Extensões do Protótipo

- **Integração com hardware real**: Implementar integração com sensores e atuadores físicos utilizando plataformas como ESP32, Arduino ou Raspberry Pi, validando a arquitetura em ambiente com dispositivos reais.
- **Autenticação e autorização**: Integrar sistema de login com JWT ou OAuth2, incluindo controle de acesso baseado em papéis (administrador, operador, visualizador), essencial para ambientes de produção.
- **Backend próprio**: Desenvolver API REST própria para persistência de configurações (regras, dashboards personalizados) e auditoria de ações, complementando os componentes FIWARE.
- **Notificações externas**: Implementar integração com serviços de email (SendGrid, AWS SES) e/ou push notifications para alertas críticos, permitindo notificação proativa para operadores, mesmo quando não estão visualizando o dashboard.
- **Aplicativo mobile**: Desenvolver aplicativo nativo (React Native) para monitoramento e controle via smartphone, ampliando a acessibilidade do sistema.

5.3.2 Melhorias Arquiteturais

- **WebSockets ou mensageria para atualização em tempo real**: Substituir o uso de *polling* por WebSockets ou Server-Sent Events, garantindo uma comunicação mais eficiente e verdadeiramente em tempo real. Avaliar a adoção de uma arquitetura baseada em mensageria, utilizando ferramentas como RabbitMQ ou Apache Kafka, a fim de melhorar a escalabilidade e o desacoplamento do sistema.
- **Dashboards personalizáveis**: Permitir que usuários criem dashboards customizados, selecionando widgets e métricas de interesse através de interface *drag-and-drop*.
- **Virtualização de listas**: Implementar virtualização para renderização eficiente de grandes listas de gráficos e entidades, melhorando performance com muitas entidades.
- **Cache inteligente**: Implementar camada de cache (Redis) para reduzir carga no Orion e QuantumLeap em cenários de alto volume de consultas.

5.3.3 Análise Avançada de Dados

- **Machine Learning**: Implementar modelos de ML para:
  - Predição de tendências (previsão de ocupação baseada em histórico)

  - Detecção de anomalias (sensor com comportamento anormal)
  - Otimização automática de thresholds baseada em padrões históricos

- ❑ **Relatórios automatizados**: Geração automática de relatórios periódicos (diário, semanal, mensal) com estatísticas e insights sobre operação do sistema.
- ❑ **Análise preditiva**: Utilizar dados históricos para identificar padrões e antecipar problemas antes que ocorram.

5.3.4 Expansão de Domínios

- ❑ **Outros contextos urbanos**: Adaptar sistema para:
  - Gestão de tráfego e estacionamentos inteligentes
  - Monitoramento ambiental urbano (qualidade do ar, ruído)
  - Iluminação pública adaptativa
  - Gestão de resíduos urbanos
- ❑ **Integração multi-domínio**: Desenvolver plataforma unificada integrando múltiplos domínios de Cidade Inteligente em um único sistema de gestão.

5.3.5 Validação e Testes

- ❑ **Testes com usuários reais**: Conduzir estudos de usabilidade com administradores de eventos e operadores de segurança para avaliar interface e funcionalidades.
- ❑ **Testes de carga**: Realizar testes sistemáticos de performance com centenas ou milhares de dispositivos simulados, identificando limites de escalabilidade.
- ❑ **Implantação em produção**: Publicar sistema em plataforma de nuvem (AWS, Azure, GCP) com domínio público e certificado SSL, validando em ambiente real.
- ❑ **Testes automatizados**: Implementar suíte de testes unitários, de integração e end-to-end para garantir qualidade e facilitar manutenção.

5.3.6 Pesquisa Acadêmica

- ❑ **Comparação de plataformas IoT**: Realizar estudo comparativo entre FIWARE e outras plataformas IoT (AWS IoT, Azure IoT Hub, ThingWorx) em critérios de performance, custo, facilidade de uso e escalabilidade.
- ❑ **Análise de segurança aprofundada**: Conduzir análise sistemática de vulnerabilidades e propor arquitetura de segurança robusta para implantação em ambientes críticos.

- **Estudo de caso longitudinal**: Acompanhar implantação do sistema em ambiente real por período prolongado, documentando desafios operacionais e benefícios observados.

Referências

BLECHMANN, S. et al. Open source platform application for smart building and smart grid controls. **Automation in Construction**, v. 145, p. 104622, 2023.

CITRON, J. W. R. **Desenvolvimento de Aplicações para Cidades Inteligentes baseado na Plataforma InterSCity**. 2025. Trabalho de Conclusão de Curso, Departamento de Computação, Universidade Federal de São Carlos, São Carlos/SP, Brasil (Bacharelado em Engenharia de Computação). Disponível em: <https://repositorio.ufscar.br/items/f0a2cfdc-04a9-4385-a6f3-a03ff28ad8e7>.

Del Esposte, A. M. et al. Interscity: A scalable microservice-based open source platform for smart cities. In: **6th International Conference on Smart Cities and Green ICT Systems**. [S.l.: s.n.], 2017.

FIWARE FOUNDATION. **FIWARE: the open source platform for our smart digital future**. 2025. Acesso em: 4 jun. 2025. Disponível em: <https://www.fiware.org/>.

FORTINO, G. et al. Internet of things as system of systems: A review of methodologies, frameworks, platforms, and tools. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, v. 51, n. 1, p. 223–236, 2021.

IBM. **O que são cidades inteligentes (smart city)?** 2023. Acesso em: 10 jul. 2025. Disponível em: <https://www.ibm.com/br-pt/topics/smart-city>.

IMPAGLIAZZO, C. et al. A testbed platform to support an iot city lab. **IoT**, v. 5, n. 1, p. 35–57, 2024.

JOSé, R.; RODRIGUES, H. A review on key innovation challenges for smart city initiatives. **Smart Cities**, v. 7, n. 1, p. 141–162, 2024.

LAI, C. S. et al. A review of technical standards for smart cities. **Clean Technologies**, v. 2, n. 3, p. 290–310, 2020.

MENEXIS, A. N. et al. Supervisory control for smart manufacturing using fiware. In: **IEEE 3rd Industrial Electronics Society Annual On-Line Conference (ONCON)**. [S.l.: s.n.], 2024.

PATEL, P.; CASSOU, D. Enabling high-level application development for the internet of things. **The Journal of Systems and Software**, v. 103, p. 62–84, 2015.

PERATA, J. P.; BETARTE, G. A methodological approach for the security analysis of fiware technology. **CLEI Electronic Journal**, v. 27, n. 4, 2024.

PUIG, F.; DíAZ, J. A. R.; SORIANO, M. A. Development of a low-cost open-source platform for smart irrigation systems. **Agronomy (Basel, Switzerland)**, v. 12, n. 12, p. 2909, 2022.

TORREPADULA, F. R. D. et al. Smart ecosystems and digital twins: An architectural perspective and a fiware-based solution. **IEEE Software**, v. 42, n. 2, p. 38–46, 2025.