UNIVERSIDADE FEDERAL DO CEARÁ

INSTITUTO UNIVERSIDADE VIRTUAL

GRADUAÇÃO EM SISTEMAS E MÍDIAS DIGITAIS

LEONARDO EVERSON LINHARES PINHEIRO

SISTEMA DE GERENCIAMENTO DE DISPOSITIVOS MQTT

FORTALEZA

2019

LEONARDO EVERSON LINHARES PINHEIRO

SISTEMA DE GERENCIAMENTO DE DISPOSITIVOS MQTT

Relatório Técnico apresentado ao Bacharelado em Sistemas e Mídias Digitais da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Sistemas e Mídias Digitais.

FORTALEZA

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

P72s Pinheiro, Leornado Everson Linhares.
Sistema de gerenciamento de dispositivos mqtt / Leornado Everson Linhares Pinheiro. – 2019.
51 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Instituto UFC Virtual, Curso de Sistemas e Mídias Digitais, Fortaleza, 2019.
Orientação: Prof. Dr. Leonardo Oliveira Moreira.

1. Mqtt. 2. Sistema. 3. Gerenciamento. I. Título.

CDD 302.23

LEONARDO EVERSON LINHARES PINHEIRO

SISTEMA DE GERENCIAMENTO DE DISPOSITIVOS MQTT

Relatório Técnico apresentado ao Bacharelado em Sistemas e Mídias Digitais da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Sistemas e Mídias Digitais.

Aprovada em: ___/___/_____.

BANCA EXAMINADORA

_____
Prof. Dr. Leonardo Oliveira Moreira (Orientador)
Universidade Federal do Ceará (UFC)

_____
Prof. Dr. Clemilson Costa dos Santos
Universidade Federal do Ceará (UFC)

_____
Prof. Dr. Flávio Rubens de Carvalho Sousa
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Primeiramente, agradeço aos meus familiares. Agradeço aos professores e colegas do curso Sistemas e Mídias Digitais que contribuíram para a minha formação acadêmica. Agradeço também ao meu professor e orientador Leonardo Oliveira Moreira, pela excelente orientação no decorrer do desenvolvimento deste relatório técnico.

"Tudo, desde indivíduos, grupos, comunidades, objetos, produtos, dados, serviços, processos serão conectados pela IoT" (SUNDMAEKER, 2009, p. 45)

RESUMO

"Internet das Coisas" — do inglês *Internet of Things* (IoT) — foi o termo usado para se referir a ideia de se etiquetar eletronicamente os produtos e facilitar a logística da cadeia de produção através de identificadores de rádio frequência. Alguns descrevem a *IoT* como um novo paradigma tecnológico em que uma rede global de máquinas e dispositivos que são capazes de interagir entre si. Pesquisas apontam que até 2020 poderemos ter algo em torno de 26 bilhões de dispositivos que se encaixam na categoria. Este relatório técnico tem como objetivo a demonstração do desenvolvimento do Sistema de Gerenciamento de Dispositivos MQTT (SGDMQTT). O relatório está separado nas seguintes etapas: na primeira etapa de caráter exploratório foi analisado o modo de funcionamento do protocolo MQTT e do *message broker* Aedes e foram definidas quais funcionalidades seriam integradas ao sistema. A segunda parte da pesquisa foi descritiva. Nesta fase o estudo tratou do desenvolvimento da aplicação, descrevendo processos como elaboração da arquitetura do sistema e desenvolvimento. Por fim, foi realizada a análise do desempenho da aplicação e demonstração dos resultados obtidos.

ABSTRACT

Internet of Things (IoT) was the term used to refer to the idea of electronically labeling products and facilitating the logistics of the production chain through radio frequency identifiers. Some describe IoT as a new technological paradigm in which a global network of machines and devices that are capable of interacting with each other. Research shows that by 2020 we can have something around 26 billion devices that fit the category. This technical report aims to demonstrate the development of the MQTT Device Management System (SGDMQTT). The report is separated in the following steps: in the first stage of exploratory character the mode of operation of the protocol MQTT and of the message broker Aedes was analyzed and what functionalities would be integrated to the system. The second part of the research was descriptive. At this stage the study dealt with the development of the application, describing processes as elaboration of the system architecture and development. Finally, the analysis of the performance of the application and demonstration of the obtained results was carried out.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Em 1999, Kevin Ashton usou o termo “Internet das Coisas” — do inglês *Internet of Things* (IoT) — para se referir a ideia de se etiquetar eletronicamente os produtos em uma apresentação para a Procter & Gamble (P&G), e com o objetivo de facilitar a logística da cadeia de produção através de identificadores de rádio frequência (ASHTON, 2009).

Lee (2015) descreve *IoT* como um novo paradigma tecnológico em que uma rede global de máquinas e dispositivos que são capazes de interagir entre si. É um conceito que pode ser expandido e aplicado para além de cadeias de produção e suprimento, e tem aplicação em diversos segmentos que compõem a sociedade, como: saúde, indústrias e entre outros (SUNDMAEKER, 2009). De acordo com Botterman (2009), citado por Sánchez et al. (2013), previsões recentes preveem que a *IoT* formará uma parte da Internet do Futuro, como seus dispositivos conectados superará em número os computadores e dispositivos móveis utilizados por usuários humanos por ordens de magnitude. Segundo a Gartner (2014), até 2020 poderemos ter algo em torno de 26 bilhões de dispositivos que se encaixam na categoria.

Um dos protocolos usados para comunicação de dispositivos *IoT* é o *Message Queue Telemetry Transport* (MQTT) que foi criado e desenvolvido pela IBM nos anos 90 com o propósito de vincular sensores em *pipelines* a satélites (IBM, 2017). O MQTT é um protocolo de sistema de mensagens assíncrono que desacopla o emissor e o receptor da mensagem tanto no espaço quanto no tempo e, portanto, é escalável em ambientes de rede que não são confiáveis (IBM, 2017). Desde de 2014 é um padrão aberto da *Organization for the Advancement of Structured Information Standards* (OASIS). Suas características permitem que seja aplicado em dispositivos mais simples e de baixo custo como o Arduino e o ESP8266 NodeMCU.

Existem diversas implementações de clientes e servidores MQTT em diversas linguagens de programação (SEBASTIAN RAFF, 2019). Os servidores que oferecem suporte ao protocolo MQTT podem também ser chamados de *broker* (WIKIPEDIA, 2019), e são eles que são responsáveis por receber e entregar as mensagens ou realizar quaisquer atividades ligadas ao protocolo.

O desenvolvimento das questões ligadas a permissões, tratamento de autenticação e gerenciamento de dispositivos geralmente devem ser implementadas a parte por aquele que deseja trabalhar com algum *broker* MQTT. Há diversas soluções que se propõem a realizar estes tipos de atividades, mas em geral seu uso é pago ou possui várias limitações no uso gratuito. A partir da situação apresentada, chegou-se a hipótese de que o desenvolvimento de um sistema gratuito e sem limitações que implementa as capacidades citadas anteriormente

pode facilitar o gerenciamento de dispositivos MQTT.

Este trabalho tem como objetivo principal o desenvolvimento um sistema que funcione integrado a um *broker* MQTT e forneça mecanismos para gerenciamento. Para alcançar o objetivo maior, foram definidos os seguintes objetivos específicos:

a) Examinar o modo de funcionamento do protocolo MQTT;
b) Definir um *broker* MQTT e examinar sua documentação;
c) Projetar e implementar o sistema *web* e os mecanismos que integrarão o *broker* ao sistema *web*;
d) Realizar testes de desempenho para o sistema *web* e para os mecanismos que integrarão o *broker* MQTT ao sistema.

Pretende-se demonstrar através dos objetivos delimitados todos os processos, tecnologias e conceitos que serão usados na construção deste relatório e disponibilizar ao final um produto que atinja os objetivos estabelecidos, agregue mais conhecimento sobre o assunto e que seja útil na administração de dispositivos.

2 REFERENCIAL TEÓRICO

Esta seção apresentará as tecnologias e conceitos que serão usados no decorrer deste trabalho.

2.1 Internet das Coisas

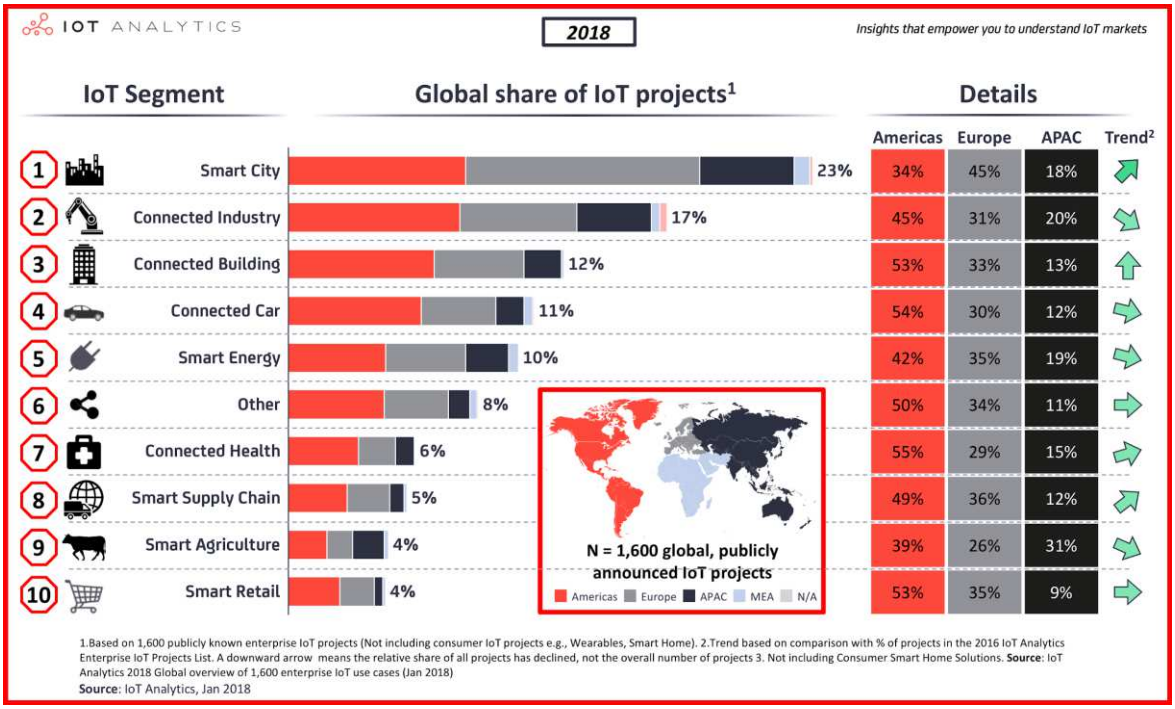
O *RFID Group* define a *IoT* como “uma rede mundial de objetos interconectados com endereços únicos baseados em protocolos de comunicação padrão” (EPOSS, 2008, tradução nossa). Lee (2015) descreve como um novo paradigma tecnológico em que uma rede global de máquinas e dispositivos que são capazes de interagir entre si. Santos et. al. (2008) descreve que a “Internet das Coisas é uma extensão da internet atual, que proporciona aos objetos do dia-a-dia (quaisquer que sejam), mas com capacidade computacional e de comunicação”. É um conceito que pode ser expandido e aplicado para além de cadeias de produção e suprimento, e tem aplicação em diversos segmentos que compõem a sociedade, como: saúde, indústrias e entre outros (SUNDMAEKER, 2009). De acordo com Botterman (2009), citado por Sánchez et al. (2013), previsões recentes preveem que a *IoT* irá formar uma parte da Internet do Futuro, como seus dispositivos conectados superará em número os computadores e dispositivos móveis utilizados por seres humanos em ordens de magnitude.

Segundo a Gartner (2014), até 2020 poderemos ter algo em torno de 26 bilhões de dispositivos que se encaixam na categoria. Sundmaeker diz que:

> "Na IoT, espera-se que "coisas" se tornem participantes ativos em diversos processos, onde eles são capazes de interagir e se comunicar entre si e com o ambiente, trocando dados e informações "percebidas" sobre o meio ambiente. Ao reagir de forma autônoma aos eventos do "mundo real/físico" e influenciá-lo, executando processos que disparam ações e criam serviços com ou sem intervenção humana direta" (2010, p. 43, tradução nossa)

A Figura 1 exibe um levantamento feito pela *IoT Analytics* sobre projetos por segmento de aplicação/região no ano de 2018. Para cada segmento *IoT* é possível notar sua influência nas diferentes regiões.

Figura 1 - Projetos de IoT por Segmento de Aplicação/Região



Fonte: IoT Analytics[1] (2018)

Torres, Rocha e Souza (2016) descrevem que "os protocolos para a comunicação entre os componentes da IoT devem lidar com fatores como baixa largura de banda, alta latência e instabilidade da comunicação". Existem alguns protocolos que são usados para fornecer os

[1] Disponível em:< https://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects/>. Acesso em 01 jun. 2019.

dispositivos a integração a Internet das Coisas, são eles: CoAP (*Constrained Application Protocol*), MQTT (*Message Queue Telemetry Transport*), XMPP (*Extensible Messaging and Presence Protocol*), *Advanced Messaging Queuing Protocol* (AMQP) e *Data Distribution Service* (DDS).

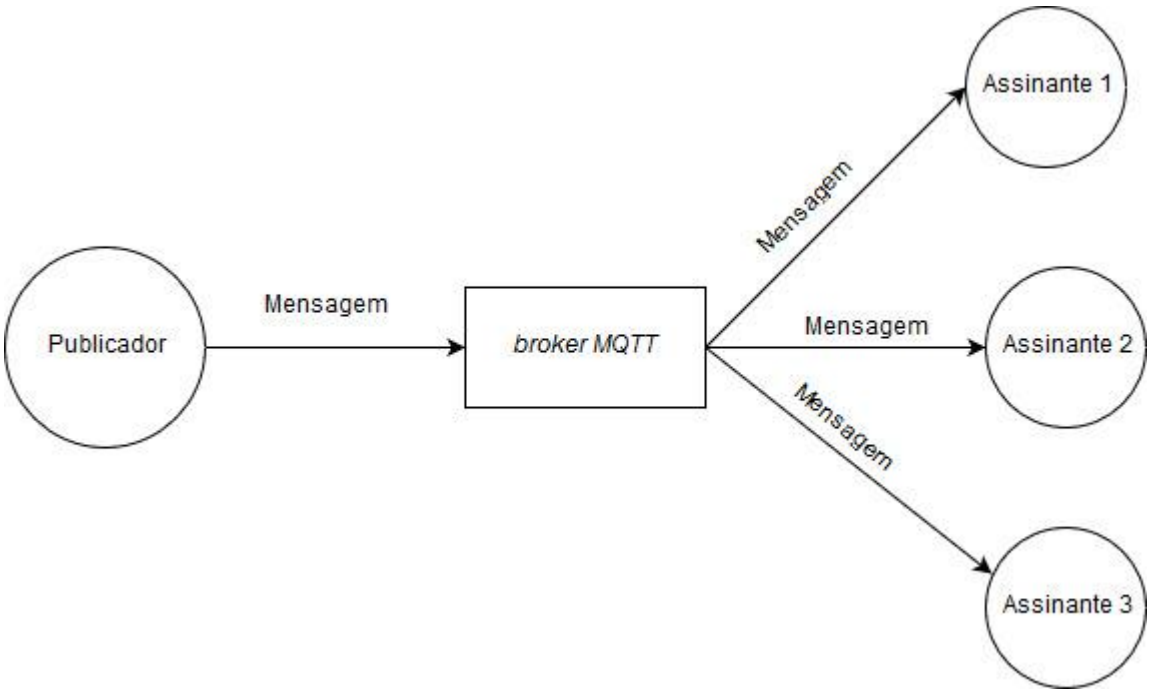
2.2 MQTT

O MQTT foi criado e desenvolvido pela IBM nos anos 1990 com o propósito de vincular sensores em dutos de petróleo a satélites (IBM, 2017). É baseado no modelo de publicação e assinatura. O MQTT é "um protocolo de sistema de mensagens assíncrono que desacopla o emissor e o receptor da mensagem tanto no espaço quanto no tempo e, portanto, é escalável em ambientes de rede que não são confiáveis" (IBM, 2017). É um padrão aberto (OASIS, 2014). Suas características permitem que seja aplicado em ambientes que possuem os fatores citados por Torres, Rocha e Souza (2016) e em sistemas embarcados dotados de pouca memória, baixo poder de processamento e limitações de energia como o Arduíno e o ESP8266 NodeMCU.

O MQTT possui suporte entre várias linguagens de programação e existem várias implementações de clientes e servidores gratuitos que oferecem suporte ao protocolo (SEBASTIAN RAFF, 2019). Para o Node.js, há duas bibliotecas MQTT que tem maior destaque na plataforma: Mosca[2] e Aedes. As duas foram desenvolvidas, principalmente, pelo mesmo autor. Neste trabalho o foco será a biblioteca Aedes, que segundo seus desenvolvedores na sua página no GitHub[3], seu design é melhorado em comparação ao Mosca. As ferramentas que fornecem suporte do lado do servidor ao protocolo MQTT podem também ser chamadas de *message broker* (WIKIPEDIA, 2019), e são elas que são responsáveis por receber e entregar as mensagens ou realizar quaisquer atividades ligadas ao protocolo. Diversas empresas que fornecem aplicações em nuvem que trabalham com o protocolo (INTERNET OF THINGS WIKI, 2019).

Figura 2 - Esquema de Funcionamento do MQTT

[2] Disponível em:< https://github.com/mcollina/mosca >. Acesso em 01 jul. 2019
[3] Disponível em:< https://github.com/mcollina/aedes/issues/190#issuecomment-372591938>. Acesso em 01 jul. 2019



Fonte: Elaborado pelo autor

Mecanismos de permissões, tratamento de autenticação e gerenciamento de dispositivos geralmente devem ser implementadas a parte por aquele que deseja trabalhar com algum *broker* MQTT. Estes são os mecanismos que serão implementados em uma aplicação web.

2.3 Web

Segundo a W3C (2019) "a *World Wide Web* (WWW, ou simplesmente Web) é um espaço de informações em que os itens de interesse, denominados recursos, são identificados por identificadores globais chamados de *Uniform Resource Identifiers* (URI) ". Esses recursos são fornecidos por servidores web e são acessíveis via navegadores. Para a construção de uma página web pode ser utilizado HTML, CSS e JavaScript. Uma aplicação *web* pode ser a combinação de recursos da própria *web* e de alguma outra tecnologia, como Java ou Node.js, do lado do servidor que permite, inclusive, acesso a banco de dados e outros recursos que servidores *web* comuns não oferecem. Com base no conceito de internet das coisas e explorando as tecnologias *web*, Guinard e Trifa (2009) propuseram o conceito de *Web of Things* (WoT) que expõe uma "abordagem semelhante para integrar dispositivos reais à *web*, permitindo que eles sejam facilmente combinados com outros recursos virtuais e físicos" (GUINARD; TRIFA, 2009, tradução nossa). O objetivo da *WoT,* segundo a Mozilla (2018) é "criar uma Internet das

Coisas descentralizada, fornecendo *URLs* (*Uniform Resource Locato*r) na *web* para torná-los vinculáveis e detectáveis, e definindo um modelo de dados e *APIs* (*Application Programming Interface*) padrão para torná-los interoperáveis". A *WoT* usa a arquitetura *REST* para comunicação entre dispositivos. A tecnologia ainda está em processo de padronização (W3C, 2019).

2.4 Node.js e Express

O Node.js é um ambiente de execução e de desenvolvimento de aplicações em JavaScript baseado no motor V8 do Google Chrome. Tem como características ser dirigido a eventos assíncronos, ser não-bloqueante e desenhado para aplicações de rede escaláveis (NODE.JS, 2019). A escolha da plataforma se dá por suas características citadas anteriormente e por mais alguns motivos: possui uma grande comunidade em torno do projeto, tem uma vasta documentação[4] e baixa curva de aprendizado no desenvolvimento de aplicações. O seu repositório de pacotes, o NPM (*Node Package Manager)*, é composto por cerca de 350 mil pacotes (LINUX.COM, 2017). Um desses pacotes é o Express, que segundo sua documentação, é framework minimalista e flexível para construção de aplicações *web* no Node.js e ainda que fornece aos desenvolvedores um conjunto robusto de recursos e técnicas para aplicativos web e móvel. (EXPRESS, 2019). O motivo da escolha do *Express* para o projeto se dá por suas diversas características e que segundo Cantelon et al. (2014), "o *framework* oferece um sistema de visualização unificada que permite usar variados mecanismos de renderização de páginas, além de oferecer vários utilitários simples que permitem: responder com vários formatos de dados, transferir arquivos, rotear *URL*s e muito mais. ". O *framework* também possui uma grande comunidade em torno do projeto além de também possuir uma grande documentação disponível[5].

2.5 MVC

Existem diversos Padrões de Arquitetura de Softwares para o desenvolvimento de aplicações. Um dos mais comuns é o *Model-View-Controller* (MVC). Segundo Gonçalves (2007), MVC é um padrão em engenharia de software que propõe a separação de uma aplicação em três camadas: *Model*, *View* e *Controller*. Leff e Rayfield (2001) descrevem que a *View*

[4] Disponível em:< https://nodejs.org/en/docs/>. Acesso em 01 mar. 2019
[5] Disponível em:< https://expressjs.com/>. Acesso em 01 mar. 2019

mostra as informações aos usuários e junto ao *Controller*, e que é a camada responsável por processar as interações dos usuários. Segundo ainda os mesmos autores, o *Model* é a porção da aplicação que contém a representação das informações mostradas pela *View* e a lógica que altera a informação em resposta a interação do usuário. O uso do MVC ainda permite que a interface da aplicação possa ser alterada sem mudar as estruturas de dados e a lógica da aplicação.

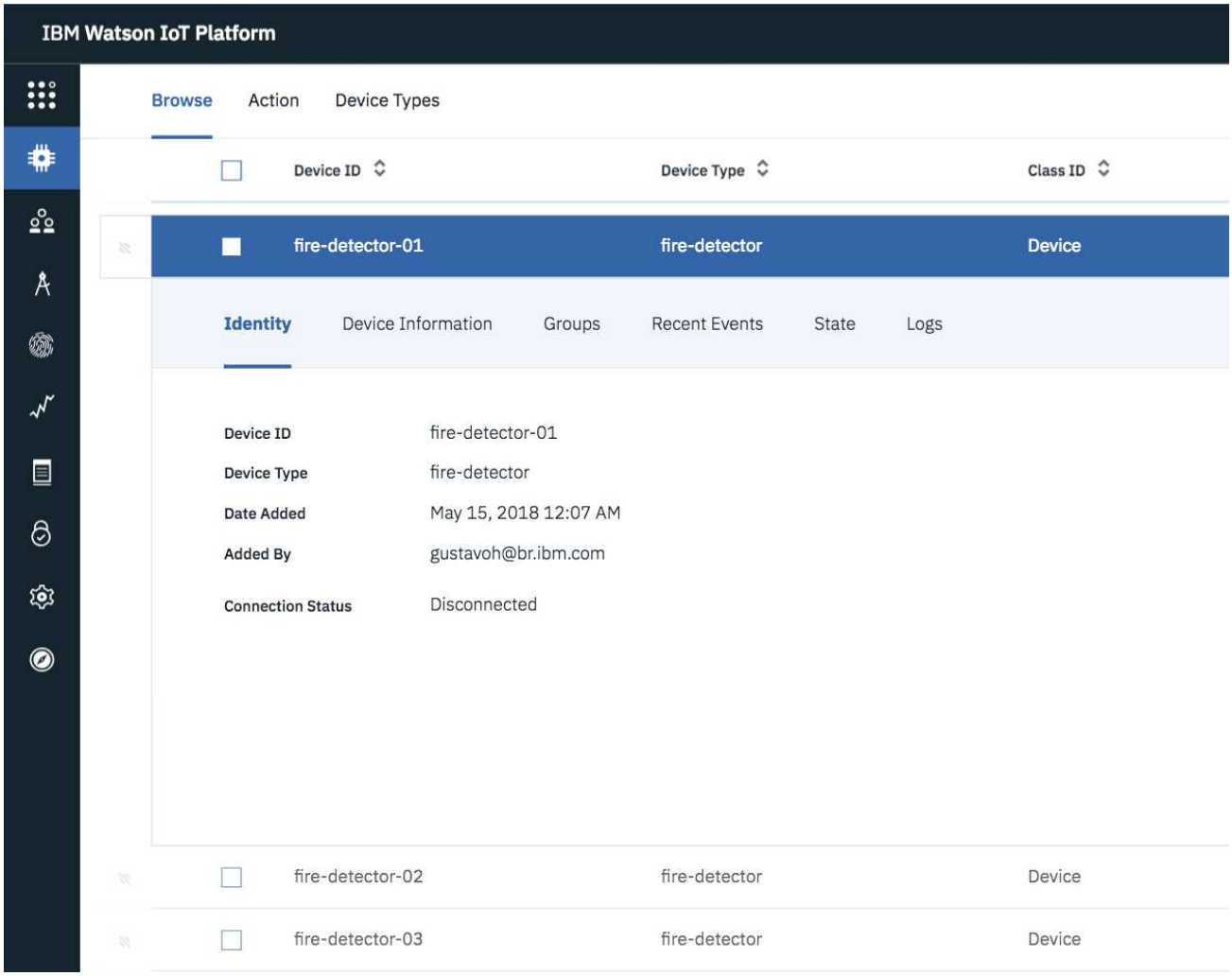
2.6 Trabalhos Relacionados

Esta parte do trabalho apresentará soluções que se assemelham ao que foi proposto como objetivo maior deste trabalho. Serão apresentadas soluções que oferecem serviço para *IoT* e destinadas ao gerenciamento de dispositivos MQTT.

2.6.1 IBM Watson IoT Platform

O *IBM Watson IoT Platform* é um serviço da IBM para gerenciamento de dispositivos MQTT na nuvem que é composto, inicialmente segundo sua documentação (IBM, 2019), pelos seguintes recursos: Uma interface web para gerenciamento de dispositivos, um banco de dados *NoSQL* Cloudant e uma instância do ambiente de execução JavaScript Node.js que roda uma instância da ferramenta Node Red. Cada instância do serviço, como exceção do banco de dados, é acessível via *URL* própria. O serviço ainda disponibiliza uma API para desenvolvedores.

Figura 3 - *IBM Watson IoT Platform Dashboard*



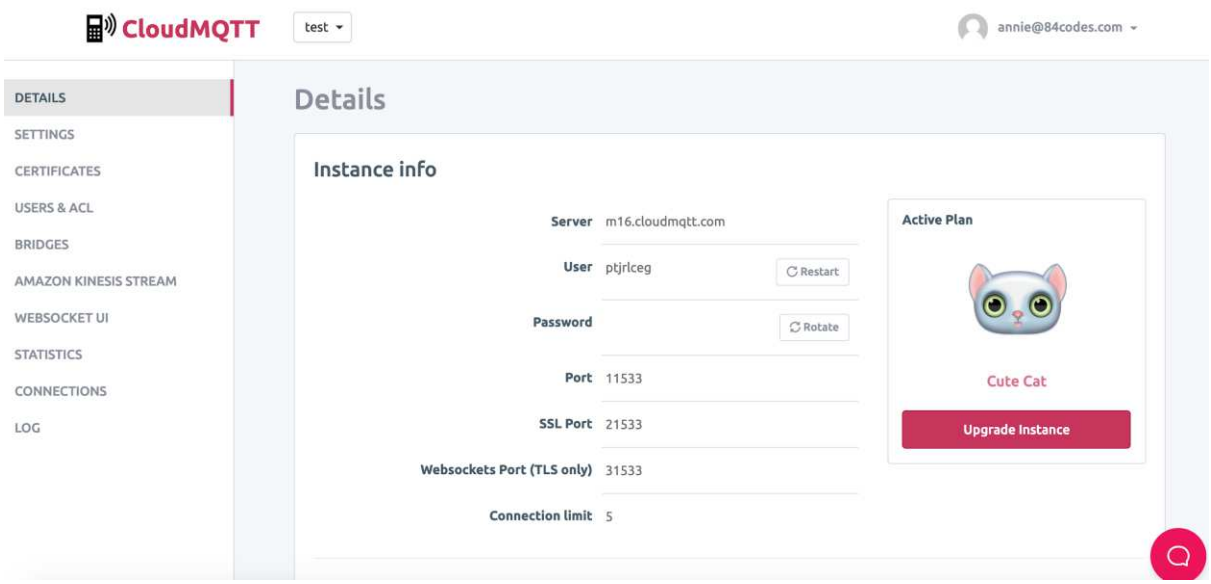
Fonte: IBM[6] (2019)

2.6.2 CloudMQTT

Cloud MQTT é um serviço que implementa instâncias do *message broker Eclipse Mosquitto* na nuvem (CLOUDMQTT, 2019). Fornece também uma interface *web* que permite acesso às informações de conexão, uma ferramenta *websocket* para publicar e se inscrever em tópicos, meios para administrar a conexão dos dispositivos e obter informações sobre estatísticas e logs da instância (CLOUDMQTT, 2019). O serviço ainda fornece uma API para desenvolvedores.

Figura 4 - Interface inicial do *CloudMQTT*

[6] Disponível em: <https://cloud.ibm.com/catalog/services/internet-of-things-platform>. Acesso em 16 mar. 2019



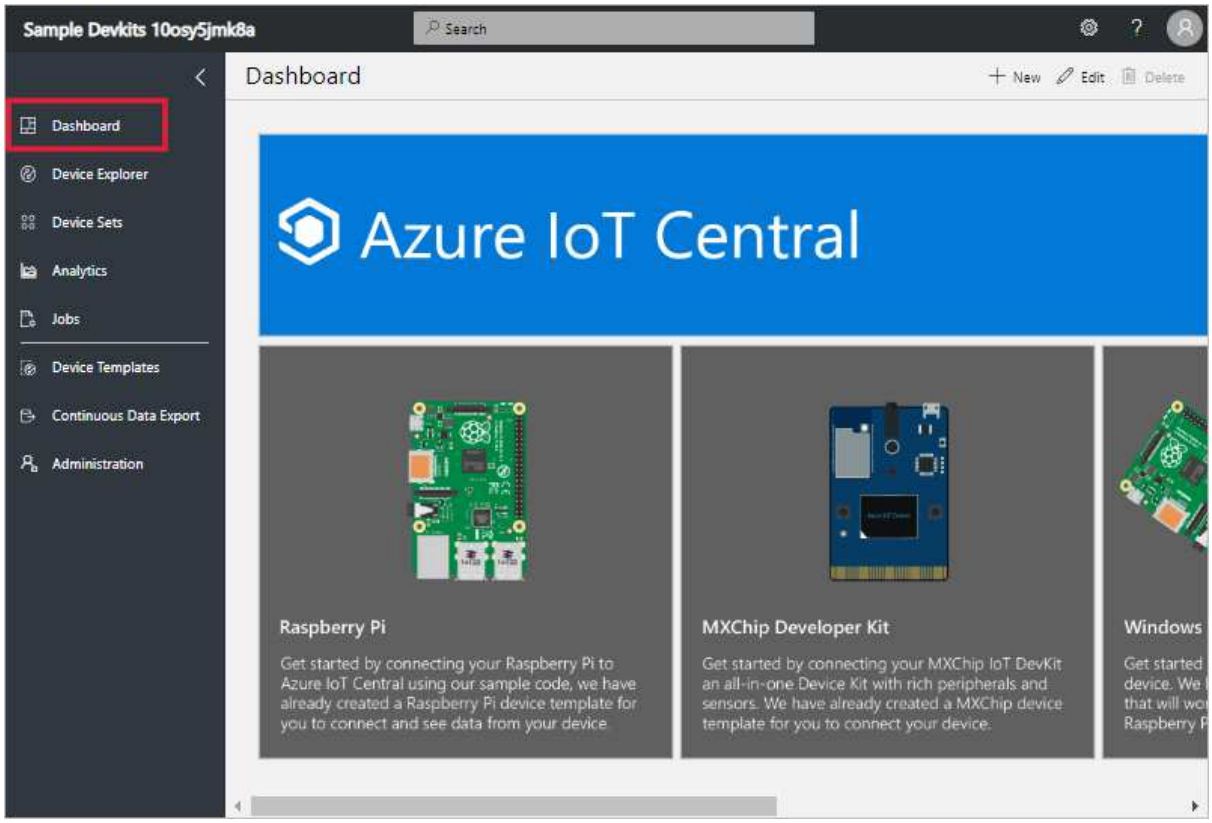
Fonte: CloudMQTT[7] (2019)

2.6.3 Azure IoT

Azure *IoT* é um conjunto de soluções da Microsoft para gerenciamento de dispositivos. Segundo sua documentação (MICROSOFT, 2018), são divididos em duas áreas: Plataforma como serviço (PaaS), que fornece soluções pré-configuradas para desenvolvimento de aplicações próprias e de aplicações *IoT* que podem usar reconhecimento contextual, e Software como Serviço (SaaS), que fornece várias aplicações para gerenciamento e monitoramento de dispositivos *IoT* (MICROSOFT, 2018). O serviço ainda possui suporte a autenticação usando certificados TLS/SSL.

Figura 5 - Página de administração do *Azure IoT Central*

[7] Disponível em: <https://www.cloudmqtt.com/docs/index.html>. Acesso em 16 mar. 2019



Fonte: Microsoft[8] (2019)

2.6.4 Google Cloud IoT

Google Cloud *IoT* é uma plataforma da Google para gerenciamento de dispositivos *IoT*. Segundo a Google (2018), A solução da empresa também "permite o processamento, análise e visualização de dados que trafegam pelo serviço". Além do protocolo MQTT, o sistema oferece a possibilidade de que dispositivos enviem seus dados através de requisições HTTP. Para segurança dos dados trafegados, o serviço dá suporte autenticação de chave assimétrica através de certificados TLS. O serviço também oferece uma API para desenvolvedores.

3 METODOLOGIA

[8] Disponível em: <https://docs.microsoft.com/en-us/azure/iot-central/overview-iot-central-tour>. Acesso em 16 mar. 2019

Inicialmente, a pesquisa tem caráter exploratório. Esse tipo de pesquisa tem como objetivo “esclarecer e modificar conceitos e ideias, tendo em vista a formulação de problemas mais precisos ou hipóteses pesquisáveis para estudos posteriores” (GIL, 2002). Nesta parte foi analisado o modo de funcionamento do protocolo MQTT e do *message broker* Aedes e quais funcionalidades seriam integradas ao sistema.

A segunda parte da pesquisa foi descritiva. Nesta fase o estudo tratou do desenvolvimento da aplicação, descrevendo processos como elaboração da arquitetura do sistema e desenvolvimento. Para implementação do sistema foi utilizada a linguagem JavaScript no *back-end* do sistema e as linguagens HTML, CSS e JavaScript no *front-end*. Inicialmente, o ambiente de desenvolvimento utilizado foi o *Sublime Text*, depois o projeto continuou a ser desenvolvido no ambiente de desenvolvimento PHPStorm. Também foi utilizada a plataforma GitHub[9] como sistema de controle de versões e para deixar público o acesso ao código fonte e ao histórico de desenvolvimento[10] do sistema. Por fim, foi realizada a análise do desempenho da aplicação.

4 AEDES

O Aedes é uma biblioteca de código aberto que oferece suporte ao MQTT no Node.js. Está disponível no GitHub[11] e pode ser instalado usando gerenciador de pacotes do Node.js, o npm. A biblioteca oferece três métodos de conexão:

a) Pela porta 1883, porta padrão do protocolo MQTT;
b) Via *websocket*, com a porta definida pelo administrador do *broker*;
c) A porta 8883, que é porta padrão do protocolo para conexões seguras.

Durante a criação de uma instância da biblioteca é possível determinar um conjunto de parâmetros opcionais:

a) ***Mq*:** relativo ao mecanismo que será responsável pela fila de mensagens. São fornecidades três opções:
   - MQEmitter;
   - MQEmitterRedis;
   - MQEmitterMongoDB.

[9] Disponível em: <https://github.com/leonardoeverson/mqtt_server>. Acesso em 01 jun. 2018
[10] Disponível em: <https://github.com/leonardoeverson/mqtt_server/commits/master>. Acesso em 01 jul. 2019
[11] Disponível em: <https://github.com/mcollina/aedes>. Acesso em 01 jun. 2018

b) ***Persistence*:** relativo ao mecanismo de persistência de mensagens. É possível usar os seguintes *softwares:*

- AedesPersistence;
- aedes-persistence-redis;
- aedes-persistence-nedb;
- aedes-persistence-mongodb.

c) ***Concurrency*:** ajusta o número máximo de distribuição de mensagens simultaneamente;

d) ***HeartbeatInterval*:** define o intervalo no qual o *heartbeat* do *broker* é emitido;

e) ***ConnectTimeout*:** configura o número máximo de ms de espera por um pacote CONNECT numa tentativa de conexão, o padrão é 30000 ms.

É possível ainda sobrescrever os seguintes métodos do programa para implementar uma lógica própria:

a) ***Authenticate*:** Método que é chamado quando algum cliente tenta autenticar com o servidor. É possível usar esse método para implementar uma lógica própria de autenticação;

b) ***AuthorizePublish*:** Função que é chamada quando algum cliente tenta publicar;

c) ***AuthorizeSubscribe*:** Função que é chamada quando algum cliente tenta se inscrever em algum tópico;

d) ***AuthorizeForward*:** Função que é chamada quando algum cliente tenta publicar;

e) ***Published*:** Função que é chamada após a publicação de mensagem.

O Aedes tem ainda os seguintes eventos definidos:

a) ***Client*:** É disparado quando algum cliente se conecta;

b) ***ClientDisconnect*:** É disparado quando o cliente desconecta;

c) ***ClientError*:** É disparado quando algum erro acontece entre o cliente e a aplicação;

d) ***ConnectionError*:** É disparado quando há um erro na conexão com o cliente;

e) ***KeepaliveTimeout*:** É disparado quando o *keepalive* do cliente expira;

f) ***Publish*:** É disparado quando um cliente publica em algum tópico;

g) ***Ack*:** É disparado quando um pacote publicado para um cliente é entregue com sucesso com QoS (*Quality of Service)* 1 ou QoS 2;

h) ***Ping*:** É disparado quando o cliente envia um *ping*;
i) ***Subscribe*:** É disparado quando um cliente se inscreve em algum tópico;
j) ***unsubscribe*:** É disparado quando um cliente se desinscreve em algum tópico;
k) ***ConnackSent*:** É disparado quando é enviado um pacote CONNACK para o cliente;
l) ***Closed*:** É disparado quando o *broker* é encerrado.

O *broker* ainda fornece dois tópicos $SYS: O $SYS/+/new/clients, que irá informar sobre novos clientes e o $SYS/+/disconnect/clients, que irá informar sobre as desconexões de clientes.

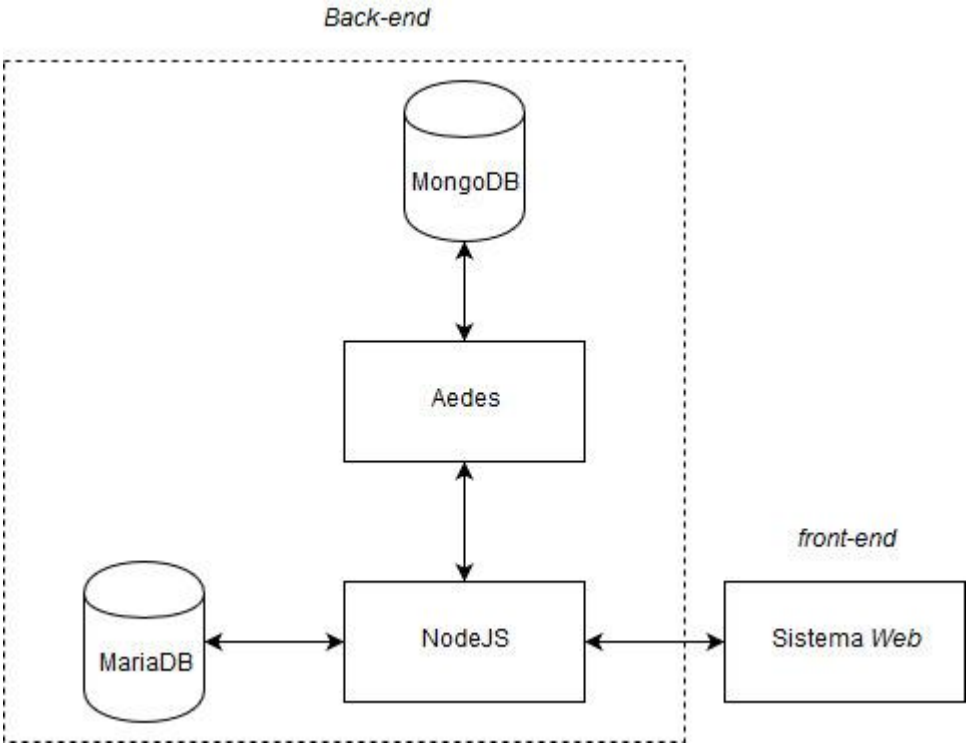
5 SGDMQTT

Esta seção tratará do planejamento e desenvolvimento de todos elementos que envolvem o objetivo maior deste trabalho.

5.1 Arquitetura do Sistema

O SGDMQTT (Sistema de Gerenciamento de Dispositivos MQTT) foi projetado para ser composto por um conjunto de camadas de software. De forma geral, a arquitetura da aplicação é cliente-servidor. O Node.js servirá duas aplicações: um sistema web, que será o *front-end* da aplicação, e a biblioteca Aedes, que será responsável pelo fornecimento de serviços relativos ao protocolo MQTT e fará parte do *back-end*. Em complemento ao *back-end*, haverá também dois bancos de dados: Uma instância do MariaDB, que servirá como persistência de dados para a aplicação *web*, e um instância do banco de dados NoSQL MongoDB, que atenderá ao Aedes como enfileirador de mensagens e persistência de dados. A definição de um banco SQL para o sistema *web* se dá principalmente pelo controle que este tipo de tecnologia oferece em como certas operações acontecem no banco de dados (ALON BRODY, 2017).

Figura 6 - Arquitetura do SGDMQTT



Fonte: Elaborado pelo autor

O Padrão MVC foi utilizado para organizar a estrutura do sistema *web*. Para complementar o padrão MVC, foi necessário ainda adicionar o mecanismo de rotas para redirecionar as chamadas HTTP para as ações corretas. Esse mecanismo será gerenciado pelo *framework* Express. Cada parte da aplicação possui uma pasta com códigos relativos à sua responsabilidade. A Figura 7 exibe parte do mecanismo de rotas relativas a função de cadastro no sistema.

Figura 7 - Rotas do arquivo cadastro.js da pasta *routes*

```
module.exports = function(app){

    app.get('/cadastro',function(request, response){
        response.render("cadastro/cadastro");
    });

    app.post('/cadastro/inserir',function(request, response){
        app.app.controllers.cadastro.cadastro_usuario(app, request, response);
    });
```

Fonte: Elaborado pelo autor

5.2 Implementação

Para a implementação do SGDMQTT, o processo foi dividido em duas partes: A construção do sistema *web* e usando JavaScript, HTML e CSS; E de mecanismos, desenvolvidos em JavaScript, que integrariam o Aedes ao sistema. O JavaScript é uma linguagem de programação dinâmica e tem como características: a relativa facilidade de uso e sua flexibilidade. É uma das mais populares linguagens de programação (TIOBE, 2019) e amplamente usada na construção de páginas *web* e que pode ser usada também para construção de aplicações no Node.js.

Para auxiliar na construção do sistema *web* foi utilizado o *framework* Express, que facilita a captura e manuseio de rotas, requisições e respostas no Node.js. O Express também dá suporte ao uso de alguns *middlewares.* No projeto foram utilizadas as seguintes dependências:

a) ***Express-session***: Mecanismo para implementação de sessões;
b) ***Express-validator***: Utilitário para tratamento de dados vindos de formulários;
c) ***Async***: Utilitário que fornece funções para trabalhar com JavaScript assíncrono;
d) ***Bcrypt***: Geração e interpretação de *hashes* de senhas;
e) ***Ejs***: Mecanismo de renderização de *templates*;
f) ***Consign***: Utilitário que facilita o carregamento de arquivos do projeto;
g) ***BodyParser***: *Middleware* que consome e analisa o corpo de uma requisição e o converte em um objeto;
h) ***Helmet***: *Middleware* que ajusta as configurações de segurança de cabeçalhos HTTP;
i) ***Mysql***: *Driver* de conexão ao MariaDB;
j) ***Uuid***: Utilitário para geração de identificadores exclusivos;
k) ***NodeMailer***: Utilitário para envio de *e-mails.*

A Figura 8 exibe todas as dependências usadas no projeto.

Figura 8 - Lista de dependências do projeto no arquivo package.json do SGDMQTT

```
"dependencies": {
  "aedes": "^0.35.3",
  "aedes-persistence-mongodb": "^5.1.0",
  "async": "^2.6.2",
  "bcrypt": "^3.0.6",
  "body-parser": "^1.19.0",
  "consign": "^0.1.6",
  "ejs": "^2.6.1",
  "express": "^4.17.1",
  "express-session": "^1.16.1",
  "express-validator": "^5.3.1",
  "helmet": "^3.18.0",
  "mongoose": "^5.5.11",
  "morgan": "^1.9.1",
  "mqemitter-mongodb": "^4.0.2",
  "mqtt": "^2.18.8",
  "mysql": "^2.17.1",
  "nodemailer": "^6.2.1",
  "npm": "^6.9.0",
  "uuid": "^3.3.2",
  "websocket-stream": "^5.5.0"
},
```

Fonte: Elaborado pelo Autor

A Figura 9 exibe as configurações usadas para determinar o funcionamento da biblioteca:

Figura 9 - Arquivo de configuração do Aedes

```
//Aedes Persistence
let persistence = aedesPersistenceMongoDB({url: url_prst});

//Emitter
let mqmongo = require('mqemitter-mongodb');

let emitter = mqmongo({url: url});

//Aedes Server
let aedes = require("aedes")({mq: emitter, persistence: persistence});

//let aedes = require('aedes')();

let server = require('net').createServer(aedes.handle);

//Servidor na porta 1883
server.listen(port, function (socket) {console.log('Servidor MQTT escutando na porta:', port)});

server.on('connection', function(client) {console.log("novo client", client.remoteAddress)});

//Servidor na porta 8888
ws.createServer({server: httpServer}, aedes.handle);

httpServer.listen(wsPort, function () { console.log('Servidor websocket escutando na porta:', wsPort)});

module.exports = aedes;
```

Fonte: Elaborado pelo autor

As Figuras 10 e 11, demonstram as configurações usadas para determinar o funcionamento do *framework* Express, do sistema *web* das dependências usadas no projeto:

Figura 10 - Configurações do *framework express*

```
let express = require('express');
let consign = require('consign');
let bodyParser = require('body-parser');
let helmet = require('helmet');
let session = require("express-session");
let morgan = require("morgan");
//let error = require("../app/middleware/error");
//console.log(process.env);

//Express
let app = express();

//body parser
//Informando ao express para usar o body parser
app.use(bodyParser.urlencoded({extended: true}));

app.use(bodyParser.json());

//app.use(bodyParser.text());

//helmet
app.use(helmet());

//Express Session
app.use(session({
    secret: '1234567890![]?:>.;@#$%¨&*()_-+§qazxswedcvfrtgbnyujmkiolpç^~;.',
    resave: false,
    saveUninitialized: false
}));
```

Fonte: Elaborado pelo autor

Figura 11 - Configurações do *framework express*

```
app.use(function(request, response, next){
    //response.setHeader("Access-Control-Allow-Origin","*")//Cross-domain
    response.setHeader("Access-Control-Allow-Methods","GET, POST, PUT, DELETE");//Cross-domain
    //response.setHeader("Access-Control-Allow-Headers","*")//Cross-domain
    response.setHeader("Access-Control-Allow-Credentials",true);//Cross-domain

    next();
});

//morgan
if(app.get('env') === 'development'){app.use(morgan('dev'))}

//Arquivos Estáticos
//Pastas estáticas
app.use(express.static('public'));

//Configurando Engine
app.set('view engine','ejs');

//Localizando Views
app.set('views','./app/views');

//Express Validator
let expressValidator = require('express-validator');

//validação de senhas
app.use(expressValidator({
    customValidators: {
        isEqual: (value1, value2) => {
            return value1 === value2
        }
    }
}));

//Localizando arquivos
//Localizando rotas e models
consign()
    .include('./app/routes')
    .then('./config/dbconn.js')
    .then('app/models')
    .then('app/controllers')
    .into(app);

module.exports = app;
```

Fonte: Elaborado pelo Autor

Para testar a integração do Aedes ao sistema conforme o sistema era desenvolvido, foram utilizados alguns Arduinos e NodeRed, que de tempos em tempos enviavam mensagens com os mais diversos parâmetros. A Figura 12 exibe parte das funções usadas nas configurações da integração entre o Aedes e o SGDMQTT

Figura 12 - Integração entre o sistema web e o Aedes

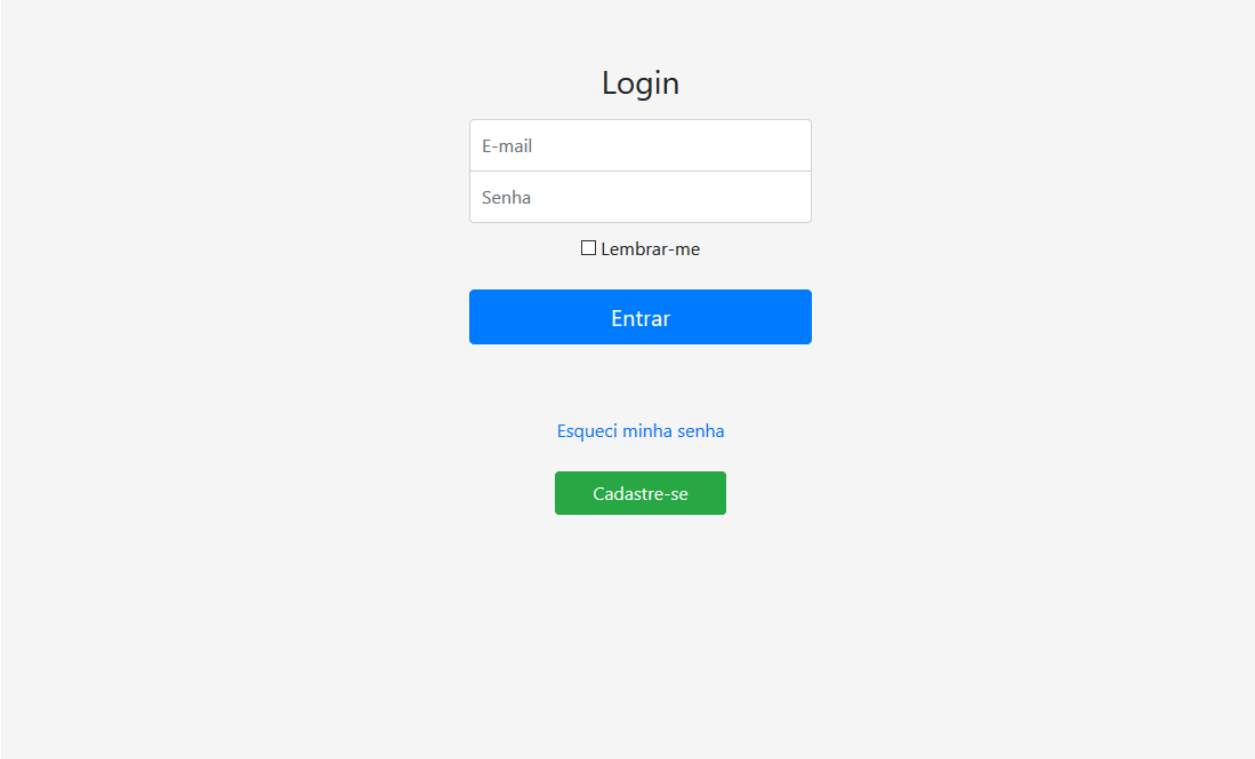
```
//Autenticação de clientes
aedes.authenticate = function (client, username, password, callback) {
    //checar novo de usuário e senha
    app.app.controllers.login.login_dispositivo(app, client, username, password, callback)
};

//Autorização de publish
aedes.authorizePublish = function (client, packet, callback) {
    app.app.controllers.topics.topic_validation(app, client, packet.topic, callback, 1);
};

//Autorização de subscribe
aedes.authorizeSubscribe = function (client, sub, callback) {
    app.app.controllers.topics.topic_validation(app, client, sub, callback, 2);
};
```

Fonte: Elaborado pelo autor

5.3 Aspectos Funcionais

Esta seção do trabalho tem como objetivo demonstrar o funcionamento do SGDMQTT. Serão demonstradas as telas e suas respectivas funções para auxiliar no entendimento. A figura 13 exibe a tela inicial da aplicação com as funções de autenticação, de cadastro e de recuperação de senha:

Figura 13 - Tela de autenticação



Fonte: Elaborado pelo autor

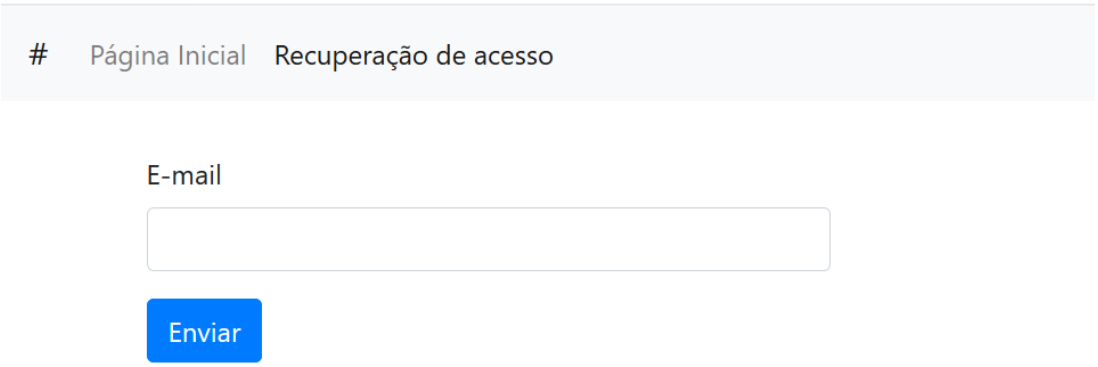
Caso o usuário não possua cadastro na plataforma, ele pode realizar o cadastro na aplicação, como demonstra a Figura 14.

Figura 14 - Tela de cadastro de usuários

Fonte: Elaborado pelo autor

A Figura 15 exibe a tela de recuperação de acesso para o usuário que esqueceu a senha. Ao inserir o *e-mail* e enviar, o sistema gerará um *link* que será enviado para o *e-mail* informado, permitindo que o usuário possa alterar a senha e recuperar o acesso ao sistema.

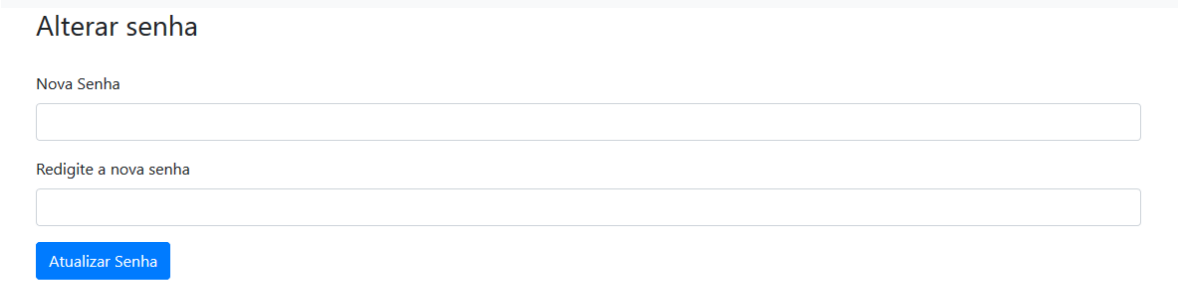
Figura 15 - Tela de solicitação de recuperação de acesso



Fonte: Elaborado pelo autor

Ao acessar o *link* enviado ao e-mail do usuário, o mesmo terá acesso a tela que a Figura 16 demonstra:

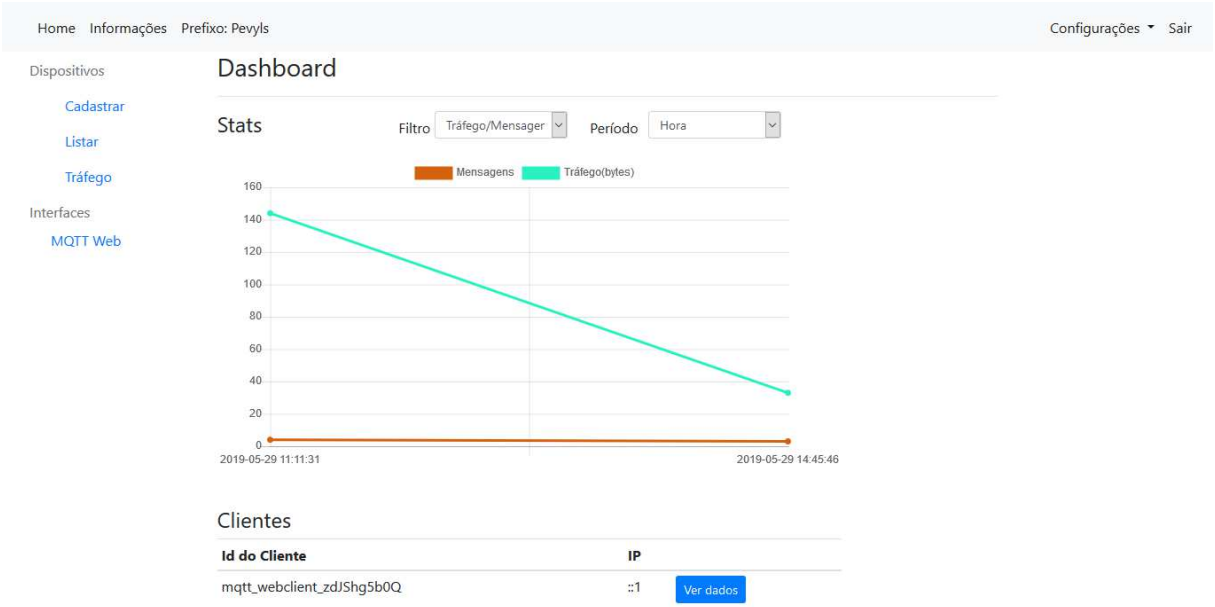
Figura 16 - Tela de recuperação de acesso



Fonte: Elaborado pelo autor

Ao inserir os dados e tentar realizar a autenticação com o sistema, caso as informações sejam válidas, o usuário será redirecionado para a tela principal mostrada na Figura 17.

Figura 17 - *Dashboard*



Fonte: Elaborado pelo autor

A tela principal é composta pelos seguintes elementos:

a) Menu superior, que se repete em todas as telas e é composto das seguintes funcionalidades:
   - *Home*: *link* que leva à página principal;
   - Informações: *link* que mostra as informações para conectar dispositivos ao *broker*;
   - Configurações: Menu *dropdown* que é composto por duas opções: Perfil e Configurações do Servidor;
   - Sair: *Link* para sair da conta atual.

b) Menu lateral que é composto dos seguintes itens:
   - Cadastrar: *Link* que redireciona o usuário a uma página de cadastro de dispositivos;
   - Listar: *link* que redireciona a uma página que lista os dispositivos cadastrados;
   - Tráfego: *link* que redireciona a uma página onde é possível visualizar o tráfego de mensagens dos dispositivos associados a conta;

- MQTT *Web*: *link* que redireciona a uma página que oferece o uso do protocolo MQTT através de uma interface *web*.

c) Gráfico central: gráfico que mostra informações de quantidade de mensagens e de *bytes* por minuto ou hora;

d) Clientes: tabela que mostra os clientes conectados e seus respectivos IP's.

Para a montagem do gráfico na tela inicial, foi usada a biblioteca chart.js que fornece um conjunto de recursos no uso de gráficos. Os dados são fornecidos através de chamadas via *Asynchronous Javascript and XML* (AJAX) para o servidor que responde no formato *JavaScript Object Notation* (JSON). A Figura 18 exibe o código usado para obter dados do servidor.

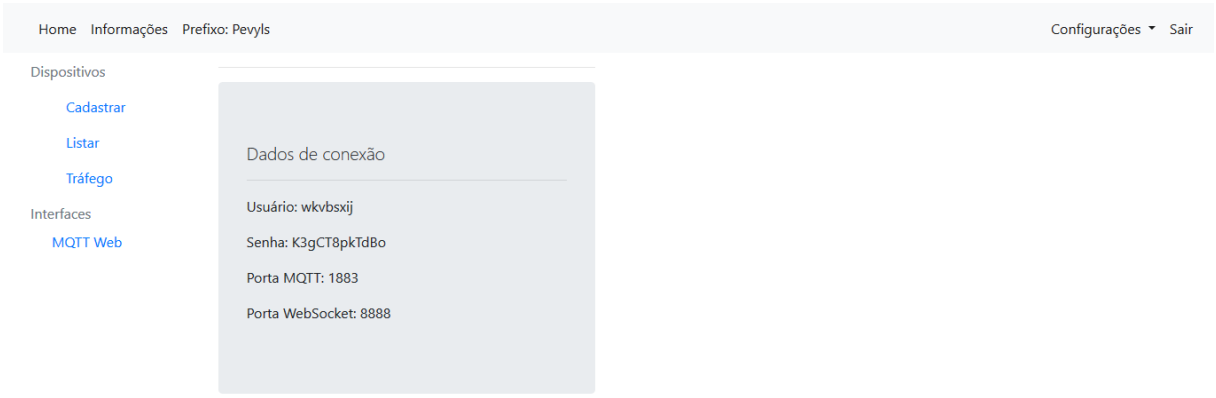
Figura 18 - Obtenção de dados do servidor

```
function carrega_dados_grafico() {

    clearInterval(timer);
    let periodo_ = periodo.value;
    let filtro_ = filtro.value;
    let xhttp = new XMLHttpRequest();

    xhttp.onreadystatechange = function () {
        if (this.readyState == 4 && this.status == 200) {
            let resposta;
            removeData(chart);
            try {
                resposta = JSON.parse(this.responseText);
            }catch (e) {
                console.log(e)
            }

            atualiza_dados_grafico(resposta);
        }
    };

    xhttp.open("GET", "/server/metrics?periodo="+periodo_+"&filtro="+filtro_, true);
    xhttp.send();
}
```

Fonte: Elaborado pelo autor

A página "Informações" mostra ao usuário os dados para conexão de dispositivos (FIGURA 19). Esses dados de usuário e senha são gerados no momento do cadastro no SGDMQTT (FIGURA 20). Esses dados são necessários para a conexão de dispositivos ao sistema.

Figura 19 - Dados para conexão de dispositivos



Fonte: Elaborado pelo Autor

Figura 20 - Geração de dados para autenticação para dispositivos

```
async function cria_prefixos(app, request) {

    let result, result1, error;

    try {
        result = await app.app.controllers.tokens.create_ids(app, request);
        result1 = await app.app.controllers.prefix.create_prefix(app, request);
    } catch (e) {
        error = e;
    }

    return new Promise(((resolve, reject) => {
        if (!error) {
            resolve([result, result1]);
        } else {
            reject(error);
        }
    }))

}
```

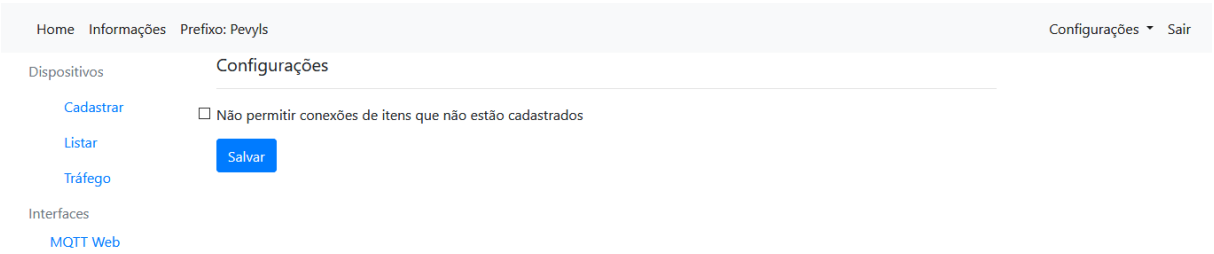
Fonte: Elaborado pelo autor.

O conjunto de caracteres após o termo “Prefixo” que se encontra no cabeçalho da página serve para informar ao usuário que ele deve usar esse termo no início dos tópicos. Esses caracteres são gerados no momento do cadastro do usuário. O uso do prefixo nos tópicos é

obrigatório e caso não se use esse termo, o servidor irá recusar a conexão do dispositivo. Essa medida foi adotada para evitar que diferentes usuários tenham acesso aos mesmos tópicos.

O menu de configurações oferece duas opções: Opções do servidor e Perfil. A opção “Opções do servidor” oferece ao usuário o poder de gerenciar a conexão de dispositivos ao *broker*, como mostra a Figura 21.

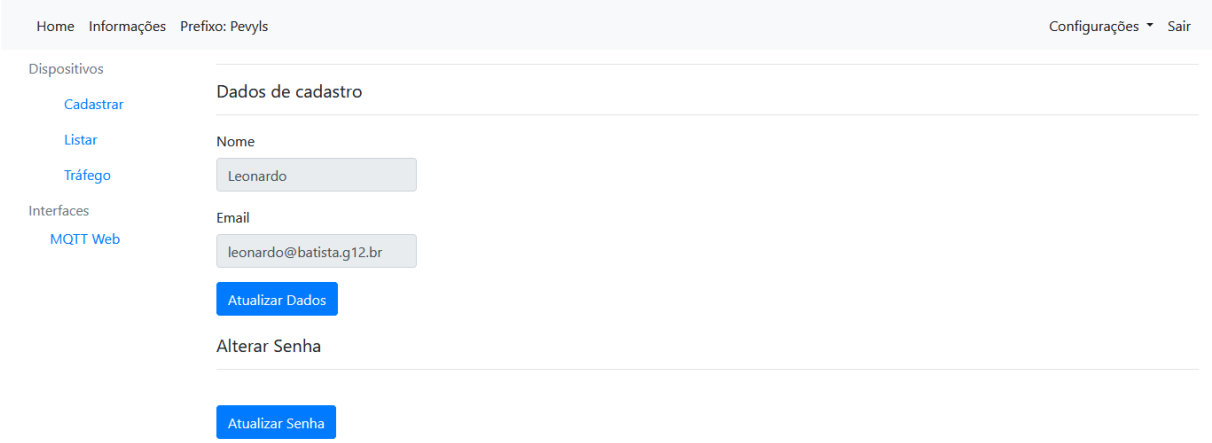
Figura 21 - Gerenciamento do Servidor



Fonte: Elaborado pelo autor.

A opção “Perfil” permite que o usuário possa editar os dados providos no momento do cadastro, como é mostrado na Figura 22. É possível alterar o *e-mail* de acesso ao sistema, seu nome e sua senha.

Figura 22 - Perfil do Usuário

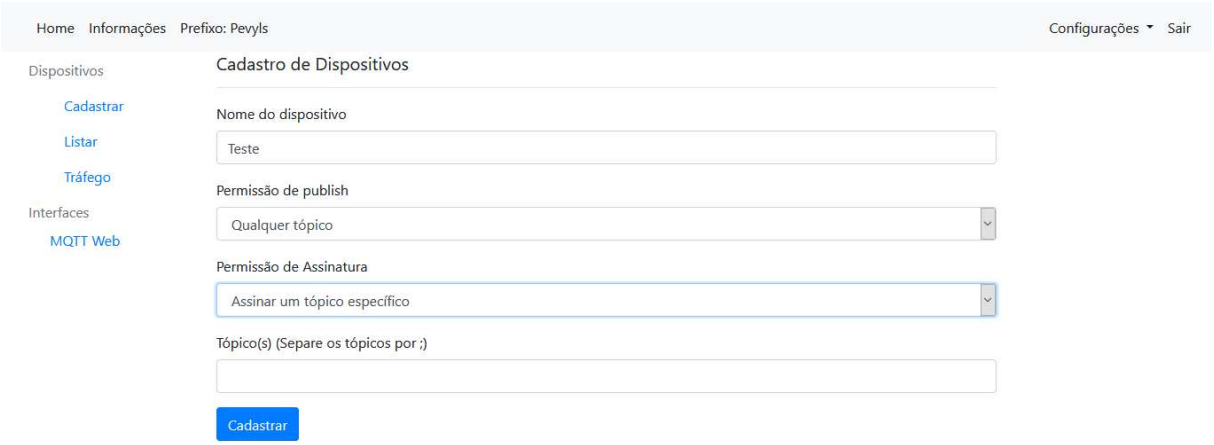


Fonte: Elaborado pelo autor

O link “Cadastro” no menu lateral do lado esquerdo leva a página de cadastro de dispositivos (FIGURA 23). Nesta página é possível definir o nome do dispositivo e definir suas permissões de publicação e inscrição em tópicos. Nas opções de publicação e assinatura, são oferecidas três opções, respectivamente, para cada:

a) Não publicar/ não permitir assinatura;
b) Publicar em tópico específico/ Assinar tópico específico;
c) Qualquer tópico.

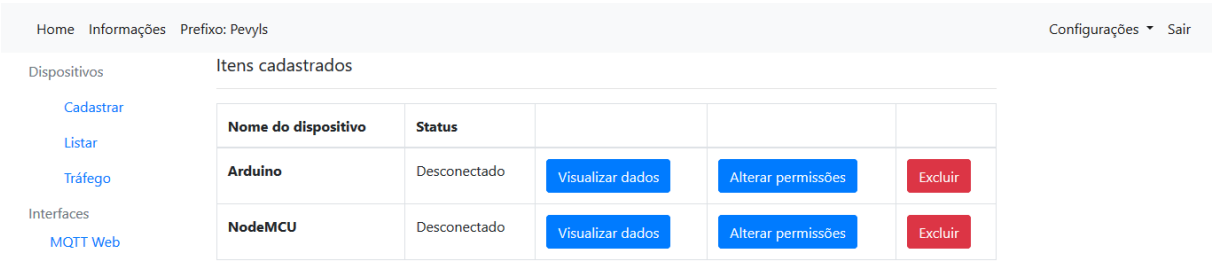
Figura 23 - Tela de cadastro de dispositivos



Fonte: Elaborado pelo autor

A página “Listar” exibe os dispositivos cadastrados (FIGURA 24), permitindo ainda a verificação do *status* de conexão, visualização dos dados que vêm dos itens, a edição de permissões e do nome do item e a exclusão do item.

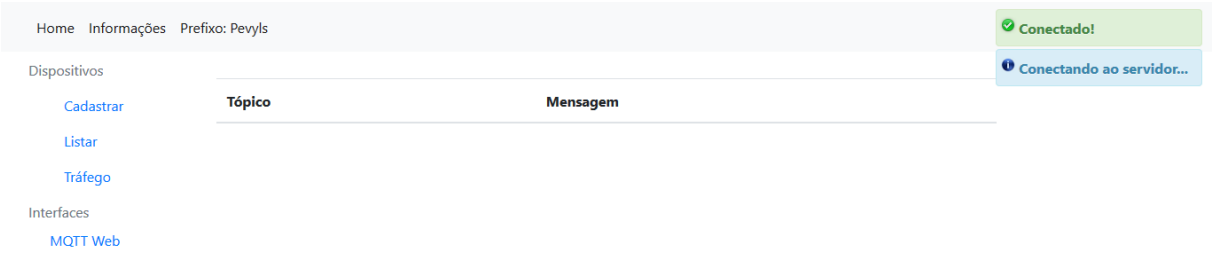
Figura 24 - Tela de itens cadastrados



Fonte: Elaborado pelo autor

O *link* “Tráfego” redireciona para a página (FIGURA 25) onde é possível visualizar todo o tráfego que passa pelo *broker* relativo aos dispositivos que estão conectados no servidor. Esse tipo de recurso é possível devido a dois fatores: O prefixo de cada usuário no sistema e o suporte do MQTT a *wildcards*. No MQTT, o mecanismo de *wildcard* permite que o usuário possa assinar a vários tópicos simultaneamente (THE HIVEMQ TEAM, 2015). O protocolo fornece suporte a dois: O de nível simples que é representado pelo sinal + e o de multinível que é representado pelo sinal #. Na tela de visualização de tráfego é utilizada a combinação do prefixo do usuário mais separador de tópico e o símbolo #.

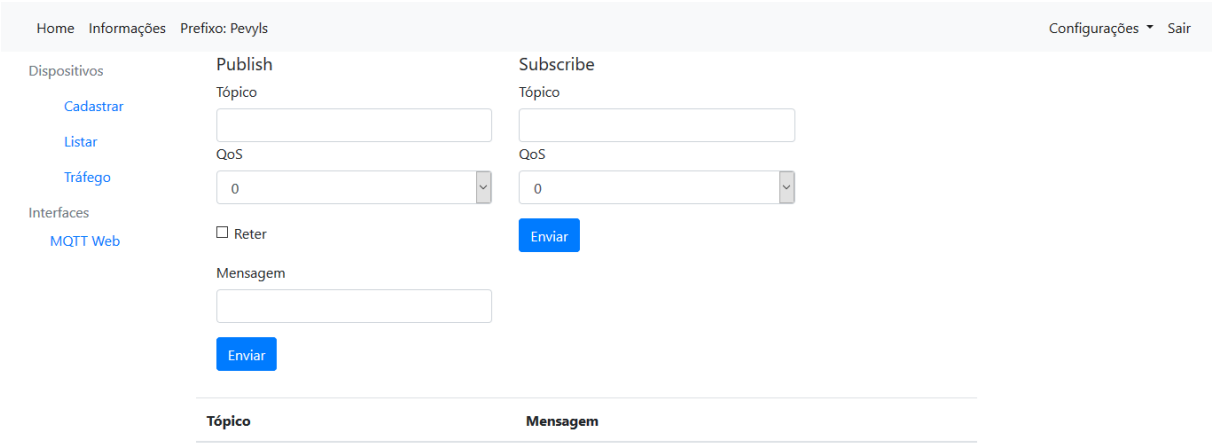
Figura 25 - Tela de visualização de tráfego



Fonte: Elaborado pelo autor

A página “MQTT Web” oferece uma interface para publicar ou assinar tópicos (FIGURA 26). Conforme as mensagens são enviadas ou recebidas do *broker*, elas são mostradas na tabela central que separa o tópico e o conteúdo da mensagem.

Figura 26 - Interface MQTT *Web*



Fonte: Elaborado pelo autor

5.4 Análise de Desempenho

Este tópico detalhará as ferramentas e configurações usadas para o desenvolvimento e execução dos testes.

5.4.1 Apache JMeter

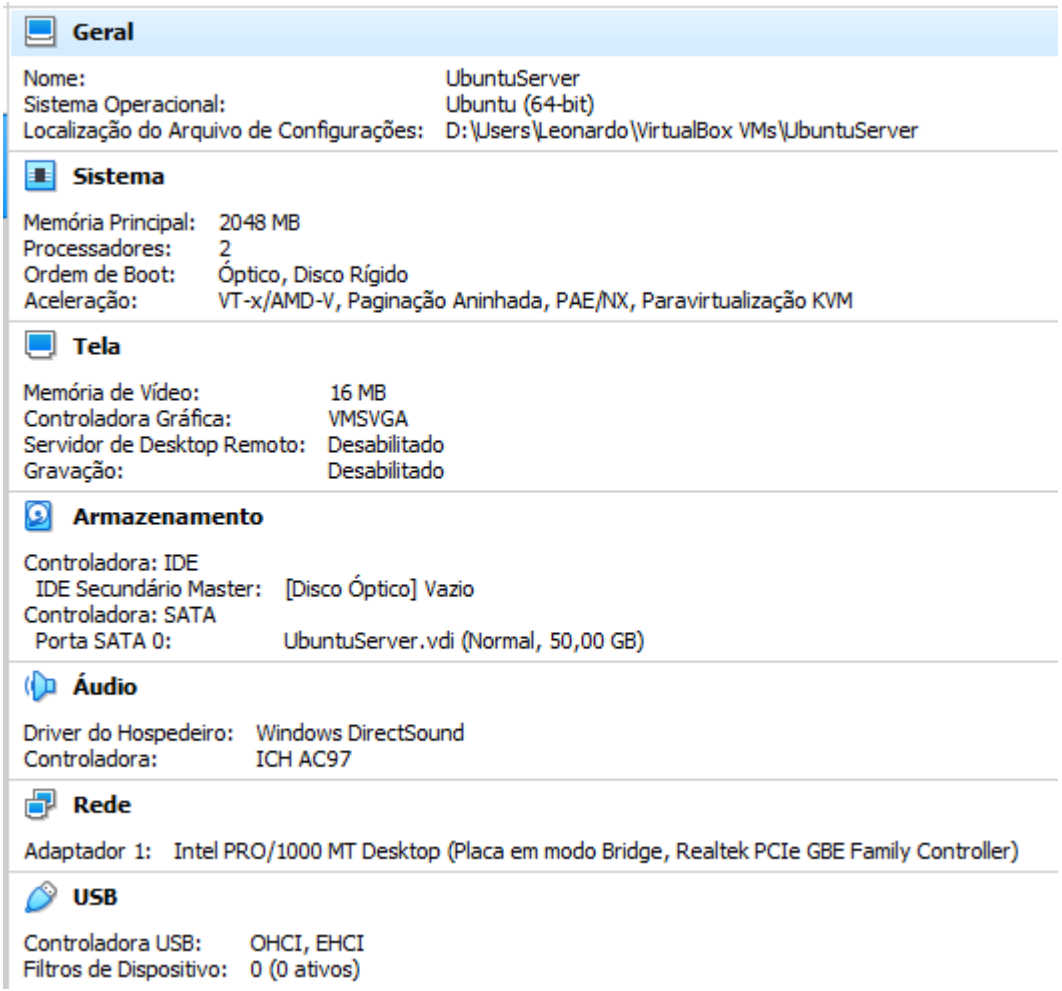
A ferramenta escolhida foi a Apache JMeter, que é uma ferramenta de código aberto, escrita em Java, que permite a construção e realização de testes de avaliação de carga e comportamento de diversas aplicações (APACHE JMETER, 2019). Permite inclusive o suporte a *plug-ins*.

5.4.2 Cenário dos Testes

Em preparação para a execução dos testes e com o propósito de ter mais controle sobre o ambiente, foram configuradas duas máquinas virtuais no programa Oracle VirtualBox: Uma máquina cliente rodando Xubuntu, uma distribuição que é baseada no Ubuntu focada na eficiência, e um servidor executando o Ubuntu Server. Ambos os sistemas se encontravam na versão 18.04.2. A máquina cliente serviu como ambiente para a execução do JMeter. O Servidor

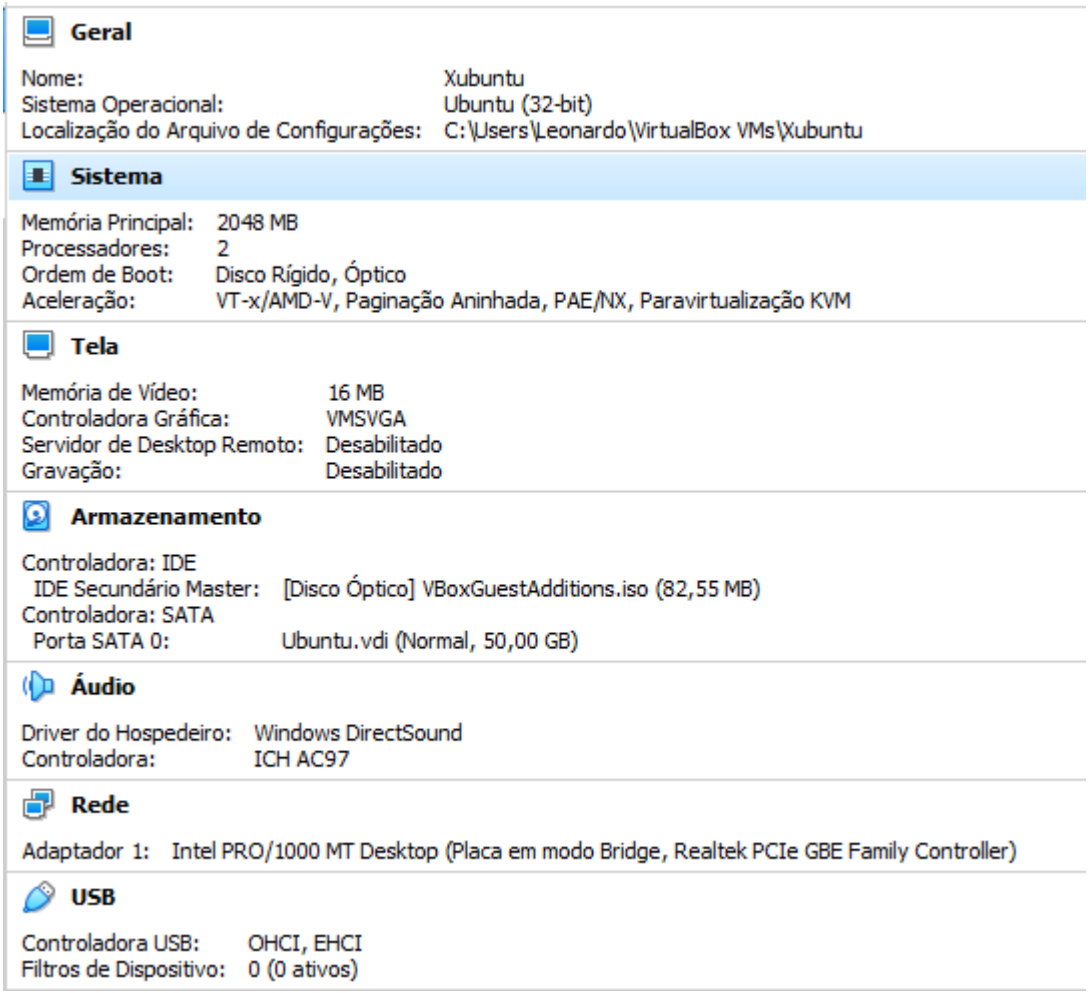
hospedou o SGDMQTT. A execução dos testes não foi possível no Windows devido a limitações em conexões de aplicativos. As Figuras 27 e 28 descrevem as configurações adotadas nas servidor e cliente, respectivamente.

Figura 26 - Configuração da máquina virtual do servidor



Fonte: Elaborado pelo autor

Figura 28 - Configuração da máquina virtual do cliente



Fonte: Elaborado pelo autor

O computador que hospedou as máquinas virtuais possui as seguintes configurações:

Tabela 1 - Descrição da máquina que hospedou as máquinas virtuais

Sistema Operacional	Windows 10 Pro Versão 1903
Memória	8 GB DDR3 1600 Mhz
Processador	Intel Core i3-4160
Placa-mãe	Gigabyte H81M-D3H
Placa de Rede	Realtek PCIe GBE 100/1000
Armazenamento	• SSD Kingston A400 500GB • HD Seagate Barracuda 1TB Modelo ST1000DM003

Fonte: Elaborado pelo autor

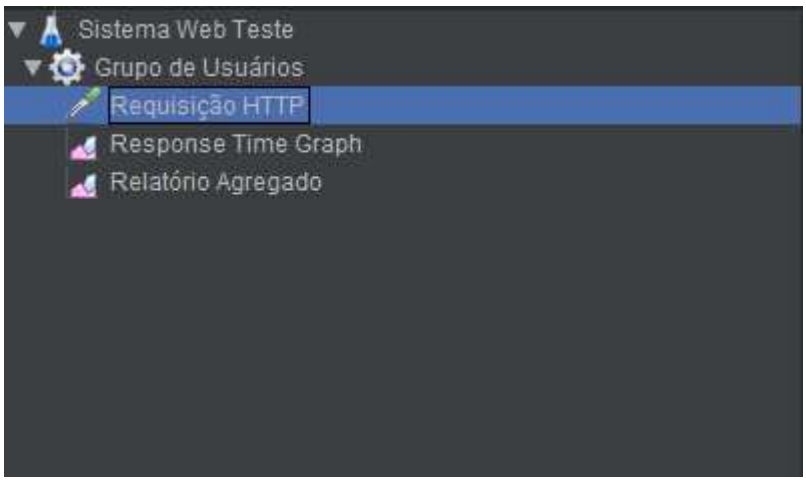
Após a instalação da máquina cliente foram executados comandos para atualizar o sistema operacional e para instalar o openjdk que é uma implementação de código aberto da plataforma Java.

No servidor também foram rodados comandos para atualização do servidor e instalação do Node.JS, MongoDB e MariaDB. Para ter acesso às últimas versões dos programas, se fez necessário usar os repositórios e documentação disponibilizada pelos responsáveis de cada sistema. No caso do *NPM*, ajustes manuais de configuração também foram feitos (NPM DOCUMENTATION, 2019).

5.4.3 Configuração dos Testes

Usando o JMeter, foram configurados dois testes: um para testar o sistema *web* e outro teste para testar o desempenho do Aedes e da integração da biblioteca com outras funções do sistema. A Figura 29 demonstra os componentes do JMeter que foram usados para realizar os testes do sistema *web*

Figura 29 - Itens do teste do sistema *web*



Fonte: Elaborado pelo autor

O elemento “Grupo de usuários” permite que se possa simular a ação de um determinado número de usuários para realizar alguma ação. É possível configurar o número de *threads,* tempo de inicialização das *threads* e configurar a quantidade de vezes que o teste irá ser executado. *Controllers*, *samplers* e *listeners* devem ficar dentro de um “Grupo de Usuários”. Como etapa inicial do teste, o componente foi configurado com o valor inicial de 50 *threads* e

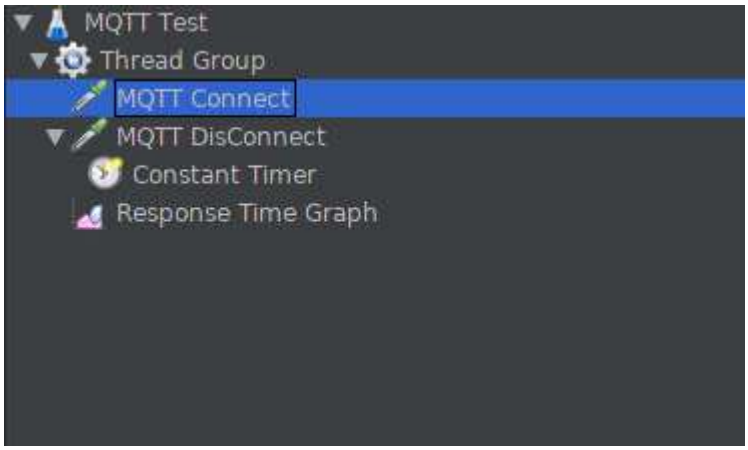
tempo de inicialização de 30 segundos. O componente “Requisição HTTP” fornece a possibilidade de realizar requisições HTTP. No componente, é possível configurar diversos parâmetros para se atingir o objetivo desejado. No caso deste trabalho foi preciso somente definir o ip do servidor e a porta. O Elemento “Response Time Graph” permite o registro em um arquivo do tempo de resposta do servidor a requisição e a visualização dos dados em um gráfico. O componente “Relatório Agregado” fornece os seguintes dados sobre as requisições:

- O número de amostras;
- A taxa de erro;
- A taxa de transferência;
- Dados enviados e recebidos;
- Média e mediana sobre o tempo de resposta das requisições;
- Exibição em porcentagem sobre o tempo de requisição.

Foram realizados 5 testes sequenciais do sistema *web* de 10 minutos. Iniciando-se com 50 usuários e a cada teste o número de usuários era incrementado em 50. O último teste foi executado com 250 usuários. A cada segundo, cada usuário fazia requisições para o sistema.

O sexto teste foi focado na integração entre o Aedes e o sistema e foi composto pelos componentes mostrados na Figura 30.

Figura 30 - Itens do teste da integração entre Aedes e o sistema



Fonte: Elaborado pelo autor

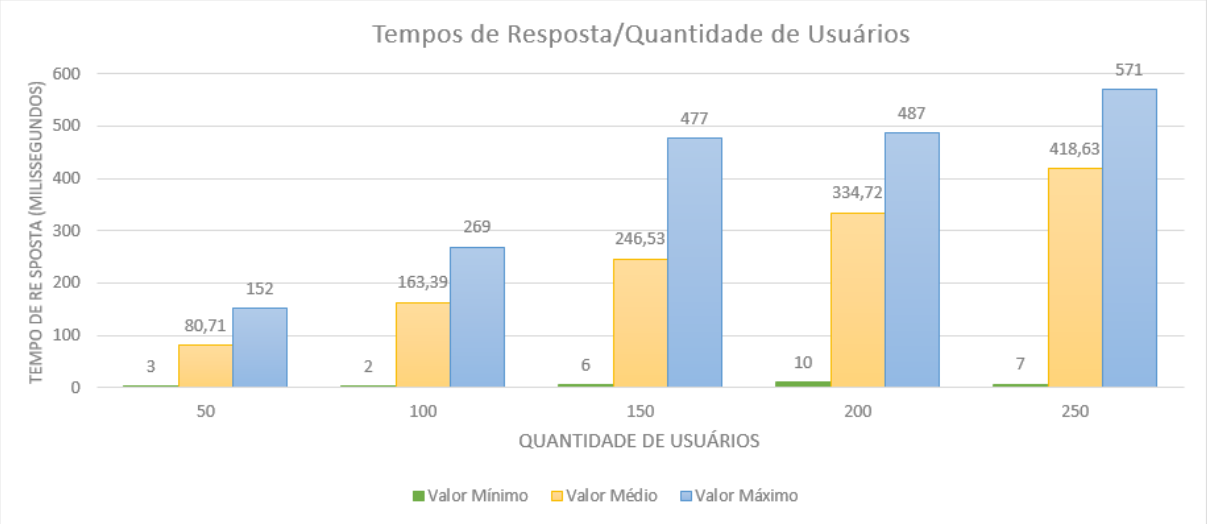
O elemento “MQTT *Connect*” permite estabelecer conexões com servidores MQTT. O elemento “MQTT *Disconnect*” permite a desconexão das conexões estabelecidas pelo MQTT *Connect.* O “Constant Timer” é um componente que controlar a execução de determinada ação ou componente. Para o teste, foi configurado que ele atrasaria o funcionamento do “MQTT *Disconnect*” em 120 segundos. O *Thread Group* (Grupo de

Usuários) foi configurado para simular 1000 usuários e com tempo de inicialização de 60 segundos.

5.4.4 Resultados

Durante os testes não foram registrados erros nas operações. A Figura 31 demonstra a evolução dos valores mínimos, médios e máximos dos tempos de resposta em cada um dos testes.

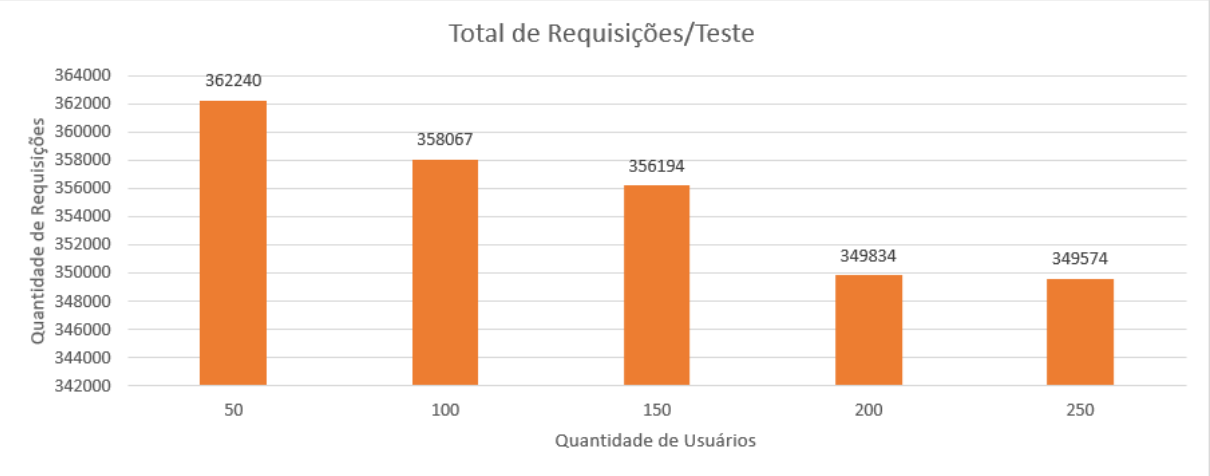
Figura 31 – Valores dos tempos de resposta por número de usuários



Fonte: Elaborado pelo autor.

A Figura 32 demonstra a mudança na quantidade de requisições em cada teste.

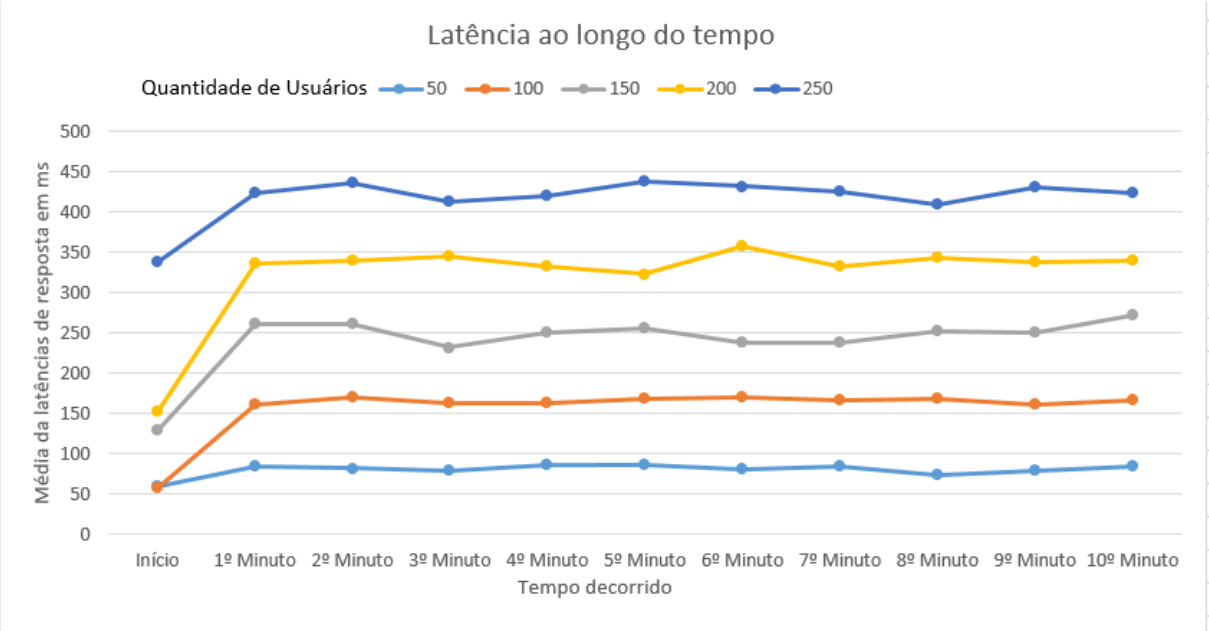
Figura 32 – Quantidade total de requisições por teste



Fonte: Elaborado pelo autor.

A Figura 33 exibe os valores médios das latências das respostas de todos os testes.

Figura 33 – Valores médios das latências das respostas por minuto



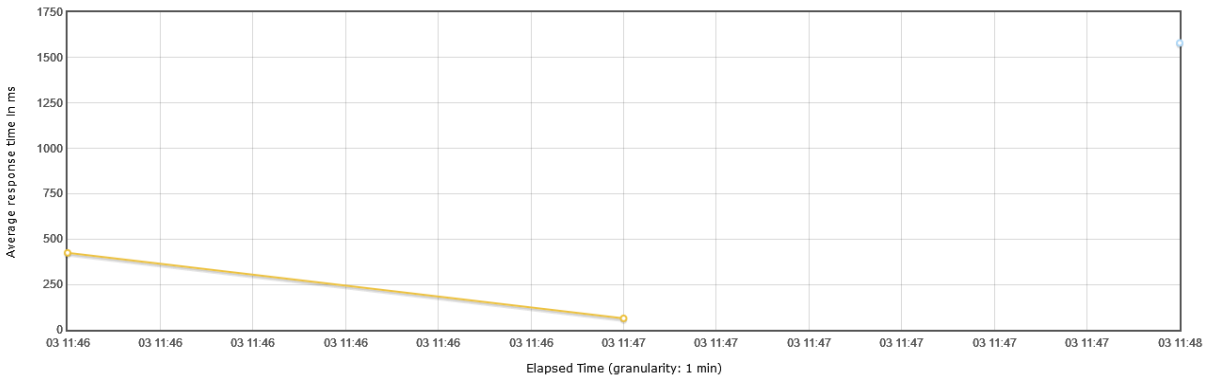
Fonte: Elaborado pelo autor.

No teste de integração entre o Aedes e o sistema foram obtidos os seguintes resultados:

- 1000 conexões/desconexões
- O tempo médio de resposta para conexões era de 140.02 ms e de 1580.74 para desconexões;
- Nas tentativas de conexão o menor tempo de resposta foi de 11 ms e o máximo foi de 10368;
- Nas tentativas de desconexão o menor tempo de resposta foi de 0 ms e o máximo foi de 4009 ms;
- O tempo de resposta máximo foi 7738 ms e o mínimo foi de 12 ms;
- Houve erro de 1.10 % no total, somando-se as operações de conexão e desconexão.
- Os erros eram de conexão não encontrada (11) e falha ao estabelecer a conexão (11).

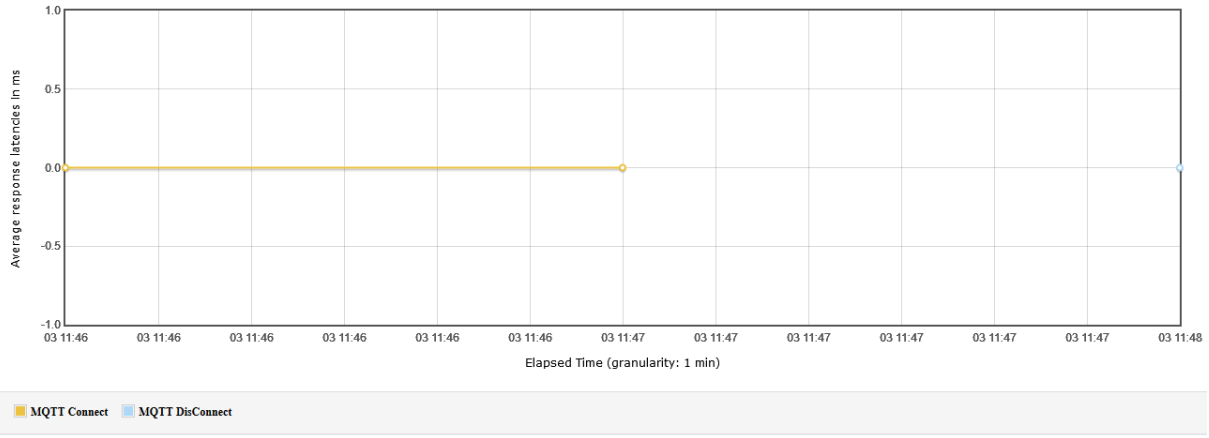
As Figuras 34, 35, 36 e 37 demonstram mais gráficos sobre o teste:

Figura 34 - Média dos tempos de resposta do sexto teste



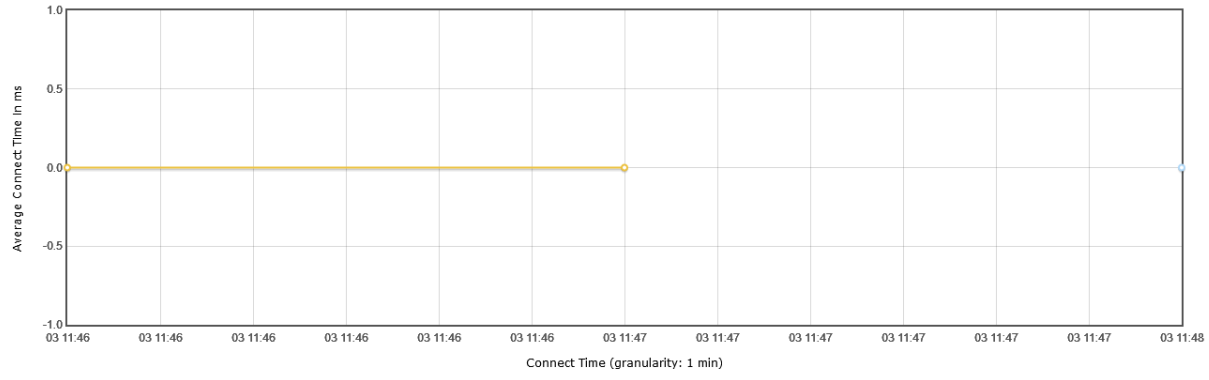
Fonte: Elaborado pelo autor

Figura 35 - Média do tempo de Latência do sexto teste



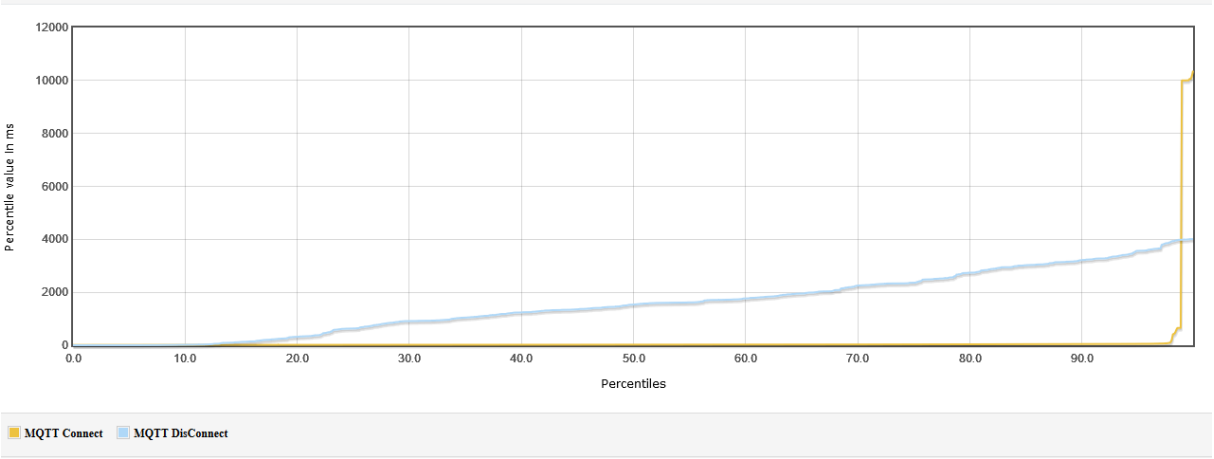
Fonte: Elaborado pelo autor

Figura 36 - Média do tempo de conexão do sexto teste



Fonte: Elaborado pelo autor

Figura 37 - Porcentagens dos tempos de resposta do sexto teste



Fonte: Elaborado pelo autor

Todos os resultados obtidos nos testes foram disponibilizados no repositório do sistema no GitHub[12].

5.4.5 Análise dos Resultados

Ao analisar os testes do sistema *web* é possível perceber a degradação no tempo de resposta e da latência conforme o número de requisições e usuários aumenta. Testes assim são necessários para encontrar problemas e implementar melhorias relativas ao desempenho do sistema conforme o número de usuários aumenta. Segundo Miguel Contreras (2017), para uma boa experiência do usuário o valor da latência de rede não deve ultrapassar 300 ms, sendo que acima desse valor, a experiência do usuário sofre degradação. Considerando esses valores, percebe-se que pode haver uma degradação na experiência do usuário a partir dos resultados obtidos no terceiro teste. É um problema que, inicialmente, pode ser solucionado em duas vias: aumento da taxa de vazão de rede e tornar o servidor escalável conforme o aumento de demanda. Por fim, é importante salientar que mesmo com o excesso de tráfego em um servidor modesto, o node.js não retornou nenhum erro nos testes.

No teste de integração entre o Aedes e o sistema, no geral, foi satisfatório. Os erros que foram detectados aconteceram na maneira como é trabalhada a conexão do sistema com o MariaDB. Foi percebido que em curto espaço de tempo são criadas diversas conexões com o banco de dados e por vezes houveram erros de *timeout*, falta de *socket* disponível e outros. Para tentar resolver o problema é necessário aumentar o número máximo de conexões permitidos pelo MariaDB e usar de um *pool* de conexões e reutilizar esse mecanismo para evitar as

[12] Disponível em:<https://github.com/leonardoeverson/mqtt_server>. Acesso em 20 jun. 2019.

constantes conexões e desconexões do sistema com banco de dados. Em ambientes críticos em que os tempos de resposta devem ser mínimos esses erros não devem acontecer e se faz necessário ainda que todos esses mecanismos sejam ainda mais otimizados. Mas ainda se faz necessário realizar um teste de carga com envio de um grande de mensagens de diversos dispositivos ao sistema.

6 CONCLUSÃO

Este trabalho teve como objetivo demonstrar o desenvolvimento de um sistema para gerenciamento de dispositivos MQTT. Para chegar ao objetivo maior, foram observadas as seguintes etapas: análise do modo de funcionamento do protocolo, definição de um *broker*, exame da documentação do *broker*, projeto e implementação do sistema. Além das etapas que levaram a implementação, também houve o teste de análise de desempenho para testar a aplicação em diversos parâmetros. Após a demonstração do resultado dos testes, foi exposta uma análise dos resultados obtidos.

Como trabalhos futuros, pretende-se: corrigir os problemas encontrados durante o teste de desempenho, realizar testes com usuários, realizar testes com dispositivos MQTT, projetar e implementar melhorias no sistema e integrar o sistema com outras funções e ferramentas que possam estender as funcionalidades e agregar mais valor ao produto.

REFERÊNCIAS

ALON BRODY. **SQL Vs NoSQL: The Differences Explained.** 2017. Disponível em: <https://blog.panoply.io/sql-or-nosql-that-is-the-question>. Acesso em: 01 jul. 2019.

APACHE JMETER. **Apache JMeter™.** Disponível em: <https://jmeter.apache.org/>. Acesso em: 02 jun. 2019.

ASHTON, Kevin et al. That 'internet of things' thing. **RFID journal**, v. 22, n. 7, p. 97-114, 2009.

BOTTERMAN, Maarten. Internet of Things: an early reality of the Future Internet. In: **Workshop Report, European Commission Information Society and Media**. 2009.

CANTELON, Mike et al. **Node in Action.** Shelter Island, NY: Manning Publications Co., 2014.

CLOUDMQTT. **Documentation | Getting started**. 2019. Disponível em: <https://www.cloudmqtt.com/docs/index.html>. Acesso em: 15 maio 2019.

EPOSS. **Internet of Things in 2020 ROADMAP FOR THE FUTURE:** INFSO D.4 Networked Enterprise & RFID INFSO G.2 Micro & Nanosystems, in: Co-operation with the Working Group RFID of the ETP EPOSS. 2008. Disponível em: <https://docbox.etsi.org/erm/Open/CERP%2020080609-10/Internet-of-Things_in_2020_EC-EPoSS_Workshop_Report_2008_v1-1.pdf>. Acesso em: 24 maio 2019.

GARTNER. **Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020**, 5 abr 2018. Disponível em: <http://www.gartner.com/document/2625419?ref=QuickSearch&sthkw=G00259115>. Acesso em: 19 mar. 2019.

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa**. 4ed. São Paulo: Editora Atlas, 2002.

GOOGLE. **Getting started**. 2019. Disponível em:<https://cloud.google.com/iot/docs/how-tos/getting-started?hl=pt-br>. Acesso em: 15 maio 2019.

IBM. **Conhecendo o MQTT**, 4 out. 2017. Disponível em: <https://www.ibm.com/developerworks/br/library/iot-mqtt-why-good-for-iot/index.html>. Acesso em: 19 mar. 2019.

IBM. **IBM Cloud e o Watson IoT Platform**. 2019. Disponível em: <https://cloud.ibm.com/docs/services/IoT?topic=iot-platform-iot-cloud-index>. Acesso em: 4 mar. 2019.

IBM. **Plataforma Internet das Coisas.** Disponível em: <https://cloud.ibm.com/catalog/services/internet-of-things-platform>. Acesso em: 26 maio 2019.

INTERNET OF THINGS WIKI. **Top 20 IoT Platforms in 2018.** Disponível em: <https://internetofthingswiki.com/top-20-iot-platforms/634/>. Acesso em: 25 maio 2019.

LEE, In; LEE, Kyoochun. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. **Business Horizons**, v. 58, n. 4, p. 431-440, 2015.

LEFF, Avraham; RAYFIELD, James T. Web-application development using the model/view/controller design pattern. In: **Proceedings fifth ieee international enterprise distributed object computing conference**. IEEE, 2001. p. 118-127.

MICROSOFT. **Soluções e tecnologias de Internet das Coisas (IoT): PaaS e SaaS**. 2018. Disponível em: < https://docs.microsoft.com/pt-br/azure/iot-fundamentals/iot-services-and-technologies>. Acesso em: 15 maio 2019.

MIGUEL CONTRERAS. **Https://www.citrix.com/blogs/2017/09/25/how-network-latency-impacts-user-experience/.** 2017. Disponível em: <https://www.citrix.com/blogs/2017/09/25/how-network-latency-impacts-user-experience/>. Acesso em: 01 jul. 2019.

MOZILLA. **Mozilla IoT - About**. 2019. Disponível em: <https://iot.mozilla.org/about/>. Acesso em: 1 maio 2019.

NPM DOCUMENTATION. **Resolving EACCES permissions errors when installing packages globally.** Disponível em: <https://docs.npmjs.com/resolving-eacces-permissions-errors-when-installing-packages-globally>. Acesso em: 01 jun. 2019.

OASIS. **MQTT Version 3.1.1 becomes an OASIS Standard**. Disponível em: https://www.oasis-open.org/news/announcements/mqtt-version-3-1-1-becomes-an-oasis-standard. Acesso em: 19 mar. 2019.

SÁNCHEZ, Luis et al. SmartSantander: Experimentation and service provision in the smart city. In: **2013 16th International Symposium on Wireless Personal Multimedia Communications (WPMC)**. IEEE, 2013. p. 1-6.

SANTOS, Bruno P. et al. Internet das coisas: da teoria à prática. **Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos**, 2016.

SEBASTIAN RAFF. **Awesome MQTT.** Disponível em: <https://github.com/hobbyquaker/awesome-mqtt>. Acesso em: 25 maio 2019.

SUNDMAEKER, Harald et al. Vision and challenges for realising the Internet of Things. **Cluster of European Research Projects on the Internet of Things, European Commision**, v. 3, n. 3, p. 34-36, 2010.

THE HIVEMQ TEAM. **MQTT Essentials Part 5: MQTT Topics & Best Practices.** 2015. Disponível em: <https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/>. Acesso em: 31 maio 2019.

TORRES, Andrei BB; ROCHA, Atslands R.; DE SOUZA, J. Neuman. Análise de desempenho de brokers mqtt em sistema de baixo custo. In: **Anais do XXXVI Congresso da Sociedade Brasileira de Computação**. 2016. p. 2804-2815.

W3C. **Help and FAQ.** 2019. Disponível em: <https://www.w3.org/Help/#webinternet>. Acesso em: 01 abr. 2019.

W3C. **Web of Things Working Group**. 2019. Disponível em: https://www.w3.org/WoT/WG/. Acesso em: 16 maio 2019

WIKIPEDIA. **Message broker.** Disponível em: <https://en.wikipedia.org/wiki/Message_broker>. Acesso em: 16 maio 2019.