UNIVERSIDADE FEDERAL DO TOCANTINS
CÂMPUS UNIVERSITÁRIO DE PALMAS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

PLATAFORMA BASEADA EM SERVIÇOS WEB PARA CONTROLE E MONITORAMENTO DE TRANSDUTORES EM AGRICULTURA DE PRECISÃO PARA IRRIGAÇÃO AUTOMÁTICA

LUCAS BERALDO ROLEDO

PALMAS (TO)
2018

LUCAS BERALDO ROLEDO

PLATAFORMA BASEADA EM SERVIÇOS WEB PARA CONTROLE E MONITORAMENTO DE TRANSDUTORES EM AGRICULTURA DE PRECISÃO PARA IRRIGAÇÃO AUTOMÁTICA

Trabalho de Conclusão de Curso II apresentado à Universidade Federal do Tocantins para obtenção do título de Bacharel em Ciência da Computação, sob a orientação do(a) Prof.(a) Me. Tiago da Silva Almeida.

Orientador: Me. Tiago da Silva Almeida

PALMAS (TO)

2018

LUCAS BERALDO ROLEDO

PLATAFORMA BASEADA EM SERVIÇOS WEB PARA CONTROLE E MONITORAMENTO DE TRANSDUTORES EM AGRICULTURA DE PRECISÃO PARA IRRIGAÇÃO AUTOMÁTICA

Trabalho de Conclusão de Curso II apresentado à UFT – Universidade Federal do Tocantins – Câmpus Universitário de Palmas, Curso de Ciência da Computação foi avaliado para a obtenção do título de Bacharel e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Data de aprovação: 8 / 6 / 2018

Banca Examinadora:

Prof. Me. Tiago da Silva Almeida

Profa. Dr. Warley Gramacho da Silva

Profa. Dr. Rafael Lima de Carvalho

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da Universidade Federal do Tocantins

R745p Roledo, Lucas Beraldo.

PLATAFORMA BASEADA EM SERVIÇOS WEB PARA CONTROLE E MONITORAMENTO DE TRANSDUTORES EM AGRICULTURA DE PRECISÃO PARA IRRIGAÇÃO AUTOMÁTICA. / Lucas Beraldo Roledo. – Palmas, TO, 2018.

53 f.

Monografia Graduação - Universidade Federal do Tocantins – Câmpus Universitário de Palmas - Curso de Ciências da Computação, 2018.

Orientador: Tiago da Silva Almeida

1. Agricultura de Precisão. 2. Internet das Coisas. 3. Servidor Embarcado. 4. Pilha MEAN. I. Título

CDD 004

A minha mãe pelo suporte, ao professor orientador pela ajuda e incentivo e aos amigos que fizeram parte dessa trajetória.

RESUMO

Este trabalho visa a criação de um servidor de aplicação embarcado para gerenciamento de dados e controle de módulos microcontrolados em uma rede de sensores sem fio. Os módulos microcontrolados são responsáveis pelo controle de transdutores em uma plantação para irrigação eficiente em agricultura de precisão, e o servidor interage com eles através de requisições do usuário e da criação de perfis de cultivo. Portanto, há a junção de duas áreas em grande expansão: Internet das Coisas e Agricultura de Precisão. O desenvolvimento de aplicações de Internet das Coisas se dá pela crescente integração de vários objetos, televisores, geladeiras, iluminação, portas etc., com serviços web. Por se tratar de um projeto extenso, o presente projeto de graduação se limita ao desenvolvimento do servidor de aplicação embarcado para gerenciamento dos dados, desenvolvido através da pilha MEAN (composto pelos frameworks MongoDB, ExpressJS, AngularJS e Node.js, respectivamente). Ele será embarcado em um Intel Edison pelo baixo custo de dispositivos SoC e baixo consumo de energia. Ao final do projeto, apresentamos o servidor como solução a ser usada no monitoramento e interação de usuários no controle automático de sistemas de irrigação de modo a se obter o uso controlado de recursos hídricos, além de discutirmos sua viabilidade.

Palavra-chave: Servidor Embarcado. Internet das Coisas. Agricultura de Precisão. Rede Wireless de Sensores. Intel Edison. Pilha MEAN.

ABSTRACT

This work aims the creation of an embedded application server for data management and control of microcontroled modules in a Wireless Sensor Network. The microcontroled modules will be responsible for the control of transducers in a plantation for efficient irrigation in precision agriculture, and the server will interact with them through user requisitions and crop profile creation. Therefore, there will be the joining of two areas in great expansion: Internet of Things and precision agriculture. The IoT applications development is due to the increasing integration of various objects, televisions, refrigerators, lighting, doors etc., with web services. Agriculture is of great interest based on the agricultural characteristic of the Tocantins state. Because it is an extensive project, the present graduation project will be limited to the development of the embedded application server for data management, developed through the MEAN stack (composed by the MongoDB, ExpressJS, AngularJS and Node.js frameworks, respectively). It will be embedded on an Intel Edison for the low cost of SoC devices and low power consumption. At the end of the project, we present the server as a solution to be used in the monitoring and interaction of users in the automatic control of irrigation systems in order to obtain the controlled use of water resources, besides discussing their viability.

Keywords: Embedded Server. Internet of Things. Precision Agriculture. Wireless Sensor Network. Intel Edison. MEAN stack.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

É de conhecimento geral que ao longo dos anos o crescimento populacional tem gerado uma necessidade cada vez maior de alimentos e com isso resultando em escalas de produção que requerem grandes quantidades de recursos naturais para suprir sua demanda, e uma das principais áreas atuantes nessa questão é a agricultura. Com isso, visando reduzir o impacto ambiental dessas atividades surge também a necessidade do uso racional desses recursos a fim de reduzir desperdícios e a ação degradante que elas acarretam, e nesse âmbito entra a Agricultura de Precisão.

A Agricultura de precisão pode ser caracterizada como a presença temporal e espacial de equipamentos e sensores que auxiliem no monitoramento do campo em relação à presença de fertilizantes, pesticidas ou irrigação (LOZOYA; AGUILAR; MENDOZA, 2016),(ABOUZAR; MICHELSON; HAMDI, 2016), (ZHOU et al., 2016). Através dela, temos a capacidade de controlar com precisão o uso de recursos e obter informações detalhadas sobre o estado do ambiente, ações que além de reduzir a degradação do meio ambiente também aumentam o lucro do produtor seja pelo aumento da produção ou pela precisão no uso de insumos.

Dentro do escopo da Agricultura de Precisão são concebidas tecnologias que permitam esse processo de coleta de dados e atuação automatizada na produção. Nos últimos anos tem havido um crescimento no desenvolvimento de projetos microcontrolados com o objetivo de oferecer serviços de hardware em aplicações móveis, sejam elas em sites da web ou aplicações de smartphones. Rapidamente, o termo Internet das Coisas foi introduzido (do inglês *Internet of Things*, ou *IoT*), em que, "coisas" representam qualquer tipo de objeto físico (não *software*) que possa ser controlado através da internet.

Esse crescimento pode ser explicado por dois fatores fundamentais: (1) a criação de um grande número de *frameworks* para desenvolvimento de aplicativos web e móveis, tais como, Angular (GOOGLE, INC, 2018a), Node.js (NODE.JS FOUNDATION, 2018a) e Bootstrap (BOOTSRAP, 2018), aumentando significativamente a facilidade e a produtividade dos projetos; (2) a criação de um grande número de plataformas microcontroladas com módulos *plug-and-play* e de projeto abertos, como por exemplo, Arduino (ARDUINO AG, 2018), Raspberry Pi (RASPBERRY PI FOUNDATION, 2018), Photon (SUPALLA, 2018) e Intel (INTEL, 2018a). As áreas de aplicação de projetos envolvendo Internet das Coisas são muitas como a engenharia, medicina, indústria entre outras, e nesse projeto damos um destaque para a agricultura, por haver uma grande demanda no estado do Tocantins por sua característica agrícola.

Por fazer parte de um sistema de irrigação automático microcontrolado, onde é feita a leitura do ambiente e do solo e tais informações são colhidas para análise e per-

sistência dos dados, este trabalho tem como parte integrante do projeto o objetivo de criar um servidor local embarcado em um módulo SoC (*System on Chip*) responsável por armazenar os dados e integrar as decisões do usuário ao sistema em campo através de uma Rede Sem Fio de Sensores (do inglês *Wireless Sensor Network*). Este tratará de um servidor desenvolvido em Node.js (uma plataforma emergente e orientada a eventos que faz uso da linguagem Javascript, conhecida no lado cliente, também no lado servidor) cujo diferencial será sua operação direta em um circuito Intel Edison que se comunicará através da rede com os demais módulos do projeto (sensores e atuadores) em um padrão de envio/recebimento de informações, dando apoio na tomada de decisões, bem como o desenvolvimento de uma aplicação web em AngularJS que permita ao usuário monitorar os dados coletados e interagir com os módulos na plantação. Por fim esperamos apresentar o servidor como solução para uma proposta de baixo custo e consumo de energia e discutir sua viabilidade como sistema embarcado e sua aplicação na irrigação microcontrolada.

1.1 IoT na Agricultura de Precisão

Uma das maiores dificuldades da agricultura e suas técnicas de produção é a sustentabilidade, cujas principais preocupações são a preservação do ambiente e o uso racional de recursos. Para contornar tais problemas, uma das principais apostas tem sido a chamada agricultura de precisão, que se baseia na contínua coleta de informações precisas do ambiente para processar a melhor ação e alocação dos recursos disponíveis. Estas informações podem incluir dados como temperatura, umidade, composição do solo dentre outras medidas que possibilitam identificar mudanças no clima, necessidade de fertilizantes e irrigação, contribuindo para a confiabilidade da produção e uma minimização de gastos.

É preciso ter em mente que a agricultura de precisão depende largamente da utilização de sistemas e ferramentas tecnológicas. É através de sensores físicos que é possível extrair informações necessárias do ambiente, e todas essas ferramentas precisam de um sistema para integrá-los e gerenciá-los, armazenando e processando as informações coletadas. Normalmente, devido a extensão dos campos agrícolas e visando suprir a necessidade de comunicação entre o sistema e dispositivos, são implementadas redes de comunicação *wireless* nestes tipos de soluções, pensadas de forma a garantir a confiabilidade no fluxo de informações onde os módulos, conhecidos como Nós ou *Nodes*, são posicionados estrategicamente em campo, permitindo a comunicação entre eles e o servidor de dados de forma eficiente (YIN et al., 2013).

Com a popular e emergente Internet das Coisas, cada vez mais dispositivos vem sendo desenvolvidos com a capacidade de interagir entre si através da Internet, facilitando a implementação e potencializando os recursos destes sistemas neste tipo de agricultura. Por exemplo, de forma ideal, se torna possível a esses sistemas se comunicarem direta-

mente com serviços de clima, fornecedores de produtos e diversas outras organizações que englobam o meio de produção agrícola. Por isso é possível ver a crescente quantidade de trabalhos que unem dispositivos IoT ao desenvolvimento de sistemas para agricultura de precisão.

Diante desse cenário, esse trabalho apresenta um servidor de aplicação executando em uma plataforma SoC desenvolvido por meio da pilha MEAN (composto pelos frameworks MongoDB, ExpressJS, AngularJS e Node.js, respectivamente). O servidor é responsável por centralizar as informações dos módulos de sensores em uma plantação para controle da irrigação automática. Os módulos são responsáveis pelo sensoriamento e atuação para irrigação. Como as mais variadas culturas precisam de irrigação e possuem demandas diferentes de água, a solução proposta utiliza-se de perfis de irrigação para culturas diferentes.

1.2 Trabalhos Relacionados

Com o surgimento da IoT, e acompanhadas de sua grande popularidade, tornou-se comum a utilização de suas tecnologias e conceitos para o desenvolvimento de soluções nas mais diversas áreas e, com isso, é possível encontrar uma gama de trabalhos científicos que validam aplicações desse tipo.

Em "*A Lightweight Platform Implementation for Internet of Things*" Heo et al. (2015), a proposta dos autores é o desenvolvimento de uma plataforma leve para Internet das Coisas, embutindo as tecnologias chaves da IoT em uma solução Node.js. Para isso, eles utilizam um nó sensor/atuador construído em um Arduino Uno R3, um servidor web desenvolvido em Node.js responsável por se comunicar com uma *RESTful* (*Representational State Transfer*, ou Transferência de Estado Representacional) API (*Application Programming Interface*, ou Interface de Programação de Aplicativos) e armazenar os dados dos sensores em uma base dados criado pelo MongoDB, um banco de dados orientado a documentos escolhido por sua estrutura de documentos JSON (*JavaScript Object Notation*, ou Notação de Objetos JavaScript) com modelos de dados dinâmicos, facilitando a integração com aplicações como o Node.js.

No artigo os autores utilizam a plataforma estabelecendo a comunicação entre os módulos através do protocolo HTTP (*HyperText Transfer Protocol*, ou Protocolo de Transferência de Hipertexto) como ocorre em páginas web, onde o método GET é utilizado para apresentar informações na API e o método POST é utilizado para transmitir dados entre os nós sensores/atuadores e o servidor, o que difere da solução apresentada no presente trabalho. Nós utilizamos o protocolo HTTP para comunicação do servidor com a aplicação do usuário, mas escolhemos o uso do protocolo MQTT (do inglês *Message Queuing Telemetry Transport*) para comunicação entre o servidor e os módulos em campo, por se tratar de um dos melhores protocolos de rede para a Internet das Coisas (IBM,

2018). Por outro lado, o artigo apresenta uma solução leve com escalabilidade para vários sensores e atuadores e que pode ser operado em *hardwares* de baixo custo indo de encontro ao objetivo deste projeto.

Em "*Using the MEAN Stack to Implement a RESTful Service for an Internet of Things Application*" Poulter, Johnston e Cox (2015) a utilização da pilha de ferramentas MEAN (MEAN.IO, 2018) na implementação de uma aplicação web IoT. É abordado o uso do Node.js, ExpressJS e MongoDB no desenvolvimento de um servidor de dados e do AngularJS em uma aplicação web, afirmando a integração e capacidade dessas ferramentas para a criação de uma solução cliente/servidor completa e altamente escalável devido a combinação do Node.js e Mongo. Também é possível ver que dentro destas plataformas podemos encontrar diversas bibliotecas de código aberto que permitem a interação direta com o hardware de um dispositivo tal como Raspberry Pi.

Em "*Design of Wireless Multi-media Sensor Network for Precision Agriculture*" (YIN et al., 2013) é apresentado o PASS (*Precision Agriculture Sensing System*, ou Sistema Sensorial de Agricultura de Precisão), baseado em uma rede multimídia sem fio de sensores composta por módulos sensores de chip único. Por não considerarem os dispositivos SoC comerciais capazes de suprir as necessidades específicas do sistema, os autores projetam um SoC chamado MSENS e a partir dele são criados os módulos sensores do PASS. Esse módulos são capazes de enviar tanto informações escaláveis (como temperatura, umidade, velocidade do vento, etc) quanto informações multimídia (como imagens da plantação para reconhecimento de doenças) e baseado nisso, devido a grande quantidade de dados gerados, é apresentado o protocolo BRDT (*Bitmap index based Reliable Data Transmission*) como um mecanismo de transmissão de dados baseado em índices Bitmap proposto para melhorar a performance da rede e transmitir as informações de forma confiável. No trabalho são apresentados os resultados dos testes em laboratório de desempenho de processamento e delay de transmissão dos dados, e também a implementação do sistema em plantações reais, na China.

Em "*Precision Agriculture Monitoring System using Wireless Sensor Network and Raspberry Pi Local Server*" (FLORES et al., 2016) é apresentado um sistema de monitoramento de agricultura de precisão que coleta dados da plantação e os envia para um servidor local executado em um Raspberry Pi. O trabalho incluí tanto sensores de temperatura, luminosidade e umidade ambiente como de umidade, condutividade elétrica e pH do solo que, ao serem coletados, enviam os dados para o servidor Raspberry responsável por processar os dados e armazená-los localmente. Esses dados são então enviados para um servidor principal em um intervalo agendado, através de um script Python que busca as informações necessárias no banco de dados local criando uma string de requisição e realiza um comando curl com essa string para enviar o conjunto de dados enquanto que no webhost existe um script PHP GET pronto para receber esses dados e armazená-los no banco de dados principal. É somente a partir desse servidor web que as informações

colhidas são apresentadas em uma interface gráfica de usuário (GUI, do inglês *Graphical User Interface*) no website desenvolvido.

Assim, desperta-se o interesse em unir estudos e tecnologias como os apresentados acima para a criação de um sistema de agricultura de precisão escalável, de baixo custo e que integre uma rede sem fio de sensores em campo agrícola.

2 SERVIDOR EMBARCADO PARA SISTEMAS DE IRRIGAÇÃO AUTOMÁTICA

A implementação de serviços em sistemas de monitoramento e gerenciamento na agricultura de precisão pode ser dividida em quatro componentes principais: módulos sensores, módulos controladores, servidores de dados e aplicações de usuário. O escopo deste trabalho se restringe ao desenvolvimento dos dois últimos componentes.

Utilizando a pilha MEAN, um conjunto de ferramentas que engloba o MongoDB, ExpressJS, AngularJS e Node.js (abordados mais detalhadamente nas Seções 2.3 a 2.5) que juntos fornecem recursos para o completo desenvolvimento de serviços tanto do servidor quanto do cliente, tornando-se uma alternativa ao conhecido LAMP (Linux, Apache, MySQL e PHP/Python) e cujo principal diferencial está no fato de que todas as ferramentas se baseiam apenas em JavaScript, já que o Node.js traz essa linguagem para o lado do servidor também, o que facilita a prototipação e desenvolvimento independente do sistema operacional utilizado, visto que a pilha MEAN pode ser instalada em Sistemas Operacionais populares como o Linux, Windows e Mac.

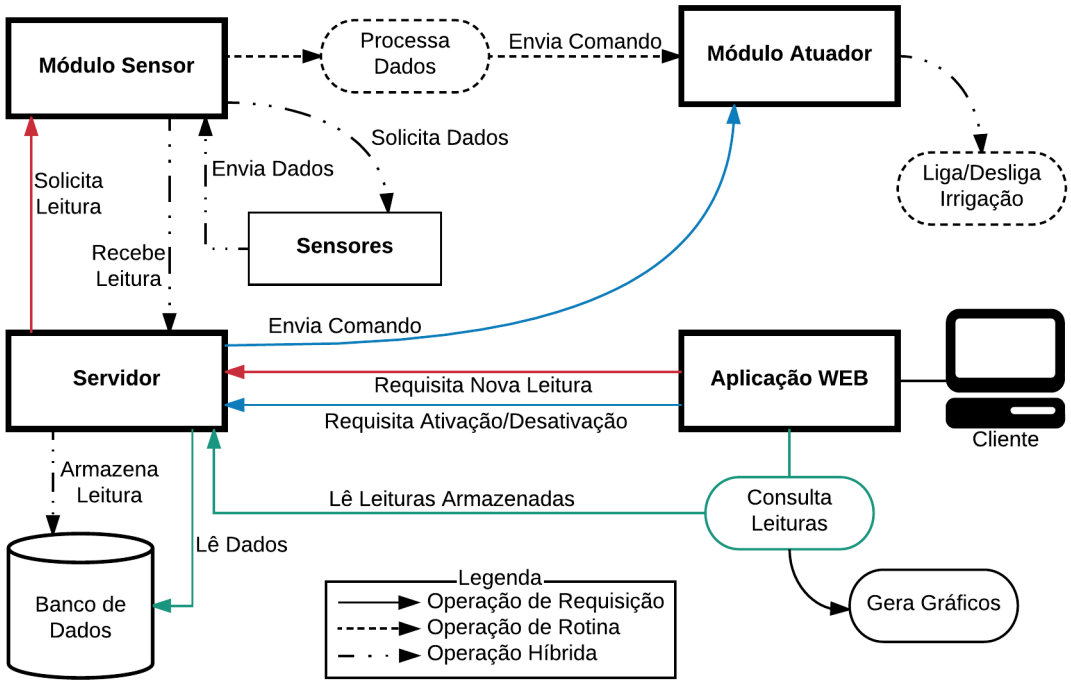
Com essas quatro ferramentas é possível criar um servidor que atuará diretamente em um dispositivo Intel e uma aplicação web que permitirá ao usuário acessar as informações e funções do sistema através da Internet. A seguir é apresentado as ferramentas e materiais utilizados no desenvolvimento da solução.

2.1 Visão Geral do Sistema

Antes de aprofundar o estudo das ferramentas e a metodologia de desenvolvimento, é importante introduzir uma visão holística do sistema de irrigação automática do qual este projeto faz parte para termos uma maior compreensão de sua operação. Além dos módulos servidor e aplicação web desenvolvidos neste trabalho, o projeto inclui um módulo sensor e um módulo atuador desenvolvidos fora do escopo desta monografia. Ainda assim, abordaremos brevemente o funcionamento de cada um e como eles interagem entre si, visualmente representados na Figura 1.

O módulo sensor é o responsável por coletar os dados que circularão pelo sistema e processá-los. Ele inclui sensores de temperatura, umidade do ar, umidade do solo, temperatura do solo, umidade da folha e de luminosidade. Desta forma, é realizada a leitura de todos os sensores conectados ao nó em um intervalo programado, e as informações obtidas são ao mesmo tempo processadas para identificar a necessidade de irrigação e enviadas ao servidor pela internet (visto na Seção 2.2) para serem armazenadas. É importante destacar que o usuário poderá solicitar, na aplicação web, uma leitura de um nó específico fora do horário programado e, neste caso, as informações são enviadas unicamente para o

Figura 1 – Visão geral da interação dos módulos do sistema de irrigação.



servidor para consulta do cliente, sem interferir na rotina dos atuadores.

O módulo atuador, ao receber as instruções do módulo de sensores, será responsável por ligar e desligar os irrigadores no campo. Essa instrução só ocorre após processamento dos dados colhidos no módulo sensor, onde é verificada a umidade atual do solo para determinar a necessidade de água na plantação, possibilitando o gerenciamento e racionalização dos recursos hídricos. Destaca-se que o usuário também poderá interagir com um módulo atuador indiretamente pela aplicação web ao enviar solicitações para ligar ou desligar os módulos, medida necessária nos casos em que o mal funcionamento de sensores possam implicar em decisões equivocadas.

O módulo servidor é o centro do sistema, sendo responsável pela persistência dos dados e por estabelecer a conexão entre o cliente e os módulos em campo. Ao receber os dados dos nós sensores, as informações são armazenadas no banco de dados do servidor. O servidor deve, então, ser capaz de retornar as consultas do usuário ao banco de dados e transmitir as requisições feitas pelo usuário.

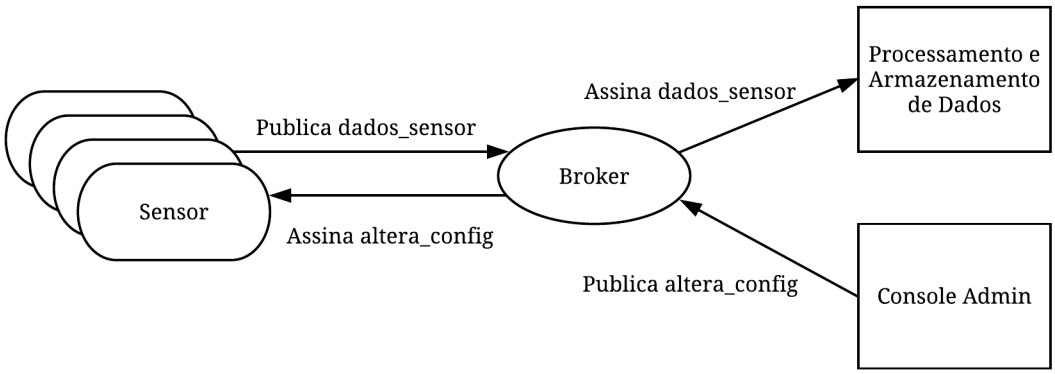
Ao final tem-se a aplicação web, que permitirá ao usuário interagir com o sistema através do navegador de internet. A partir dela, o cliente pode consultar as leituras, cadastrar perfis de plantações, cadastrar os módulos em campo e acessá-los para realizar solicitações como nova leitura, ativação/desativação do módulo e também configurá-lo remotamente.

2.2 Comunicação: Protocolo MQTT

Se tratando de dispositivos de Internet das Coisas, é requisito fundamental haver conexão com a internet. Dentro deste escopo, existem os chamados protocolos de rede e se tratando de IoT emerge como um dos principais padrões o protocolo MQTT, um protocolo de mensagens assíncrono leve e flexível que permite a implementação em *hardware* de dispositivo altamente restringido e em redes de largura da banda limitada e de alta latência (IBM, 2018).

Diferente do HTTP (*Hypertext Transfer Protocol*) que é um protocolo de um para um, o MQTT facilita e reduz o custo de transmitir uma mensagem a todos os dispositivos na rede (caso de uso comum em aplicativos de IoT) através do seu modelo de publicação e assinatura (visto na Figura 2). Neste modelo são definidas duas entidades: um *broker* de mensagens (servidor que recebe as mensagens publicadas e as roteia aos assinantes interessados) e o cliente (representando tanto dispositivos quanto aplicativos).

Figura 2 – Modelo de envio de mensagens do *broker* MQTT (IBM, 2018).



Após estabelecida a conexão do cliente, ele pode publicar mensagens em quaisquer tópicos do *broker* e, da mesma forma, pode realizar uma ou mais assinaturas a qualquer tópico passando a receber todas as mensagens publicadas nesse tópico específico, dando ao desenvolvedor a flexibilidade de definir quais clientes interagem com quais mensagens. O trecho de código da Listagem 1 referencia como é estabelecida a conexão do servidor com o *broker* e a assinatura/publicação em tópicos.

```
1 var cliente  = mqtt.connect('mqtt://m13.cloudmqtt.com:PORT',{ //Conecta
    com o broker, neste exemplo o CloudMQTT
2     username: 'usuario', //Autenticacao no broker
3     password: 'senha' //Autenticacao no broker
4 });
5
6 cliente.on("connect", function () {
7     cliente.subscribe("TOPICO"); //Assina TOPICO no broker
```

```
8     cliente.publish("OUTRO", mensagem); //Publica uma mensagem no topico
          OUTRO
```

Listagem 1 – Conexão, assinatura e publicação em MQTT.

Um *broker* de mensagem local pode ser instalado diretamente em *hardware*, tendo como mais conhecidos o Mosca e o Mosquitto, porém já existem serviços de plataforma IoT que fornecem um *broker* online e que pode ser acessado diretamente pela internet como é o caso do CloudMQTT, solução escolhida para este projeto de irrigação automática, sendo responsável pela comunicação entre o servidor e os módulos sensores em campo.

2.3 O Servidor

Neste trabalho, foi escolhido a plataforma Node.js para a programação do módulo servidor. O artigo "*Is Node.js a Viable Option for Building Modern Web Applications? A Performance Evaluation Study*" Chaniotis, Kyriakou e Tselikas (2015) aponta que um servidor implementado em JavaScript através do Node.js supera em desempenho um servidor Apache PHP tradicional. No estudo, em um servidor baseado em um Intel Core i7 920 @ 2.6GHz com 9GB DDR3 @ 1066 MHz, são realizados uma série de testes de estresse em três tecnologias populares em servidores: um baseado em Node.js com JavaScript, um Apache e um Nginx, ambos em PHP. Os resultados apontam que o servidor Node.js é de forma geral superior em performance computacional.

Em testes de E/S são feitas requisições HTTP pelos clientes no browser onde cada servidor responde com uma string "Hello Word", apresentando resultados a favor do servidor Node.js em relação ao Nginx e Apache, sendo que o Node.js atende cerca de duas vezes mais requisições que o Apache, além de exibir maior utilização da CPU (o que significa menos ociosidade dos recursos) e o melhor gerenciamento da memória durante toda a execução do teste, como pode ser visto nos gráficos da Figura 3.

São realizados também um teste computacional/*hashing* (operação comum em servidores web), onde um conjunto fixo de coordenadas geográficas é codificado em cada solicitação, sendo que o servidor Node.js fica a frente dos demais, dessa vez atendendo quase três vezes mais requisições que ambos os servidores PHP, como pode ser visto nos gráficos da Figura 4. Ele não apresenta perda significante de performance em relação ao primeiro teste, o que indica sua capacidade de lidar com o processamento de dados de forma mais eficiente que o PHP. O único teste em que o Node.js se mostrou inferior foi em servir arquivos estáticos, onde, apesar de manter a consistência no uso da CPU, foi o que apresentou maior consumo de memória durante a execução dos testes.

É importante destacar que a utilização conjunta destas quatro ferramentas da pilha MEAN dão suporte ao padrão de projetos MVC (do inglês *Model-View-Controller*, ou Modelo-Visão-Controlador) tanto no lado do servidor, através do Node.js + Express, quanto no lado cliente, através do framework AngularJS. O padrão MVC (MICROSOFT,

Figura 3 – Desempenho do Node.js em Operações de E/S baseado em conexões concorrentes (CHANIOTIS; KYRIAKOU; TSELIKAS, 2015).

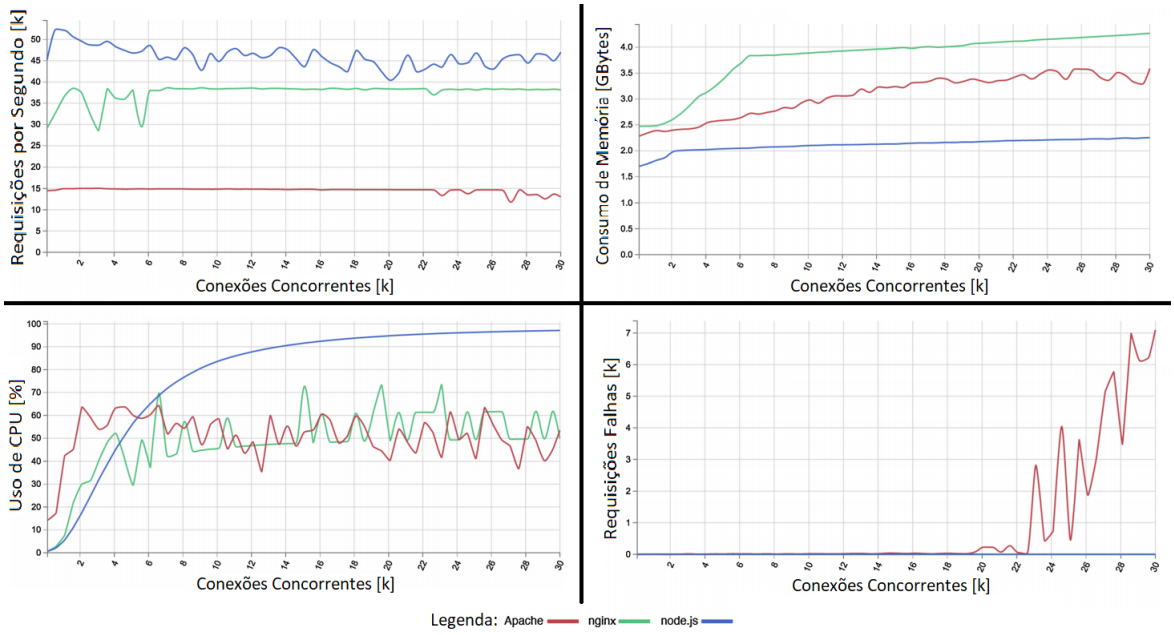
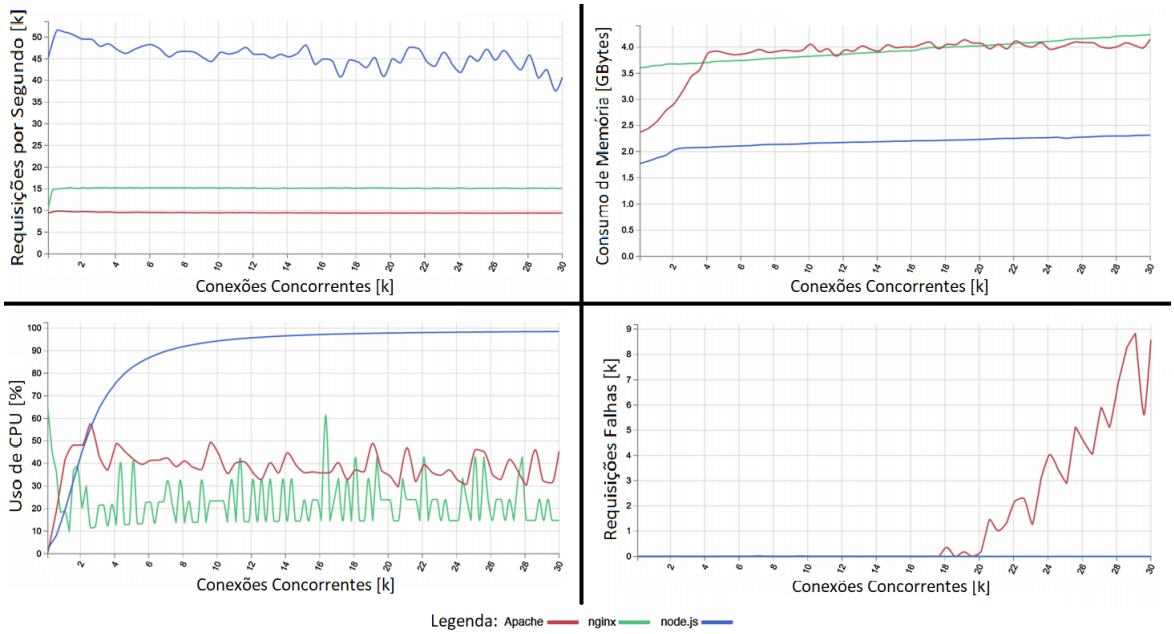


Figura 4 – Desempenho do Node.js em Operações de Hash baseado em conexões concorrentes (CHANIOTIS; KYRIAKOU; TSELIKAS, 2015).

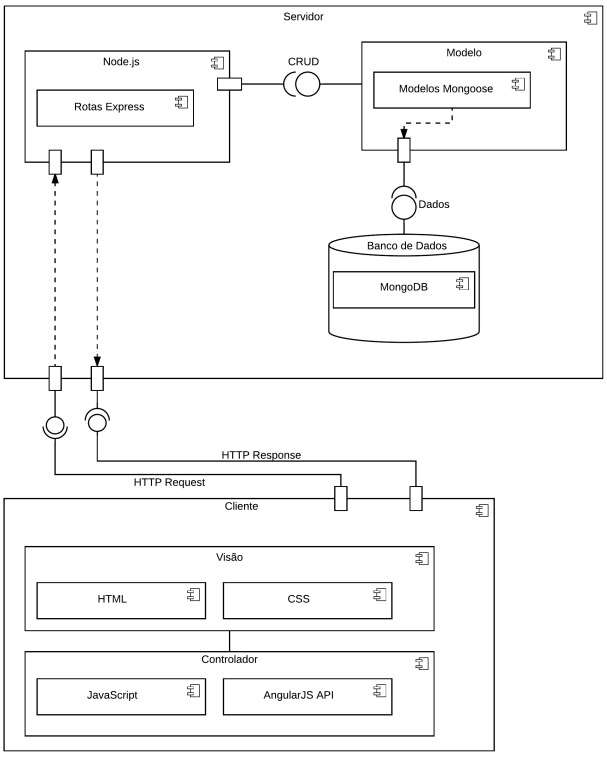


2018) é uma forma de dividir e representar uma aplicação em três camadas distintas: modelo, visão e controlador.

De forma geral, neste tipo de divisão, a camada Modelo (*Model*) é a responsável pela manipulação dos dados, o que inclui a leitura, gravação e validação dos dados que circulam a aplicação. A camada Visão (*View*) é a que interage diretamente com o usuário, sendo responsável por exibir os dados ao cliente normalmente por meio do HTML

(*HyperText Markup Language*, ou Linguagem de Marcação de Hipertexto). A camada Controlador (*Controller*) é a camada que recebe todas as requisições do usuário, traçando uma ação que decide qual *model* utilizar para gerenciar os dados e/ou qual página da *view* o usuário verá, sendo em outras palavras o responsável pela comunicação entre as demais camadas do modelo. Uma representação visual dos componentes deste trabalho que faz uso do padrão MVC pode ser vista na Figura 5.

Figura 5 – Diagrama de Componentes do projeto Cliente/Servidor.



Como o servidor será responsável por mediar as interações entre o usuário da aplicação web e os módulos sensores/atuadores em campo, é preciso garantir a comunicação e o fluxo de informações entre eles. Definimos a comunicação entre o servidor e aplicação web através do protocolo HTTP. O HTTP permite a comunicação entre cliente e servidor através de um padrão do tipo requisição-resposta (chamados *Requests* e *Responses*, respectivamente). Os principais métodos de envio de informação nesse padrão são o HTTP GET e HTTP POST.

2.3.1 Node.js

O Node.js é a principal ferramenta do conjunto, sendo utilizado no desenvolvimento de soluções cliente/servidor em um ambiente inteiramente Javascript. Construído na Engine V8 da Google, ele fornece um servidor assíncrono de alta performance orientado a eventos (*Event-Driven*). Além disso, a plataforma Node.js inclui um gerenciador de pacotes chamado *npm* (*Node Package Manager*), considerado o maior ecosistema de

bibliotecas de código aberto no mundo, permitindo uma fácil instalação e gerenciamento dos módulos e bibliotecas da aplicação uma vez que todas as dependências do projeto, referenciadas no arquivo "package.json", podem ser manipuladas por ele. Um exemplo da facilidade de utilização está no comando abaixo, que instala o módulo Express necessário no projeto sem a necessidade de passos adicionais:

```
1 npm install express --save
```

Seu principal diferencial está no fato de funcionar com um único *thread* e que sua arquitetura é completamente não-bloqueante, garantindo uma boa performance com consumo de memória e, principalmente, eliminando a existência de *deadlocks* no sistema já que os processos nunca bloqueiam. (NODE.JS FOUNDATION, 2018a)

O Node também possui nativamente o libuv (LIBUV CONTRIBUTORS, 2018), uma biblioteca multiplataforma que trabalha com operações de E/S assíncronas, lidando por "debaixo dos panos" com o *threading*, *thread pooling* etc., sem que o desenvolvedor tenha que interagir diretamente com isso. Na sua documentação é possível ver que ela possui recursos como:

- Implementação de um Event-Loop completo;
- Sockets TCP e UDP assíncronos;
- Resolução de DNS assíncrona;
- Operações de arquivos e sistemas de arquivos assíncronas;
- Processo filho (*Child Process*);
- *Thread pool*;
- *Threading* e primitivas de sincronização.

Desta forma, como as funções em Node.js são executadas de forma assíncrona, com chamadas não bloqueantes, ele faz com que tarefas sejam executadas em paralelo, o que maximiza o processamento e reduz o tempo de execução como observado nos testes de Chaniotis, Kyriakou e Tselikas (2015). Podemos observar a diferença entre a execução de processos de forma síncrona e assíncrona na Figura 6.

Uma das características do Node.js é que ele é uma plataforma orientada a eventos, da mesma forma que o Javascript funciona no lado cliente, onde a única diferença está na inexistência de eventos como *click* e *keyup*, que dão lugar a eventos de E/S no servidor, como o *connect* em banco de dados, o *open* de arquivos, entre outros. Nele, há um agente chamado de *Event-Loop* que é responsável pelos eventos do sistema, executando um loop infinito onde a cada iteração é verificado se existem eventos em uma Fila de Eventos para que o *Event-Loop* execute tal evento e retorne uma função de resposta adequada, chamada de *callback*. É possível ver o funcionamento básico do Event-Loop na Figura 7.

Figura 6 – Processos Síncronos x Processos Assíncronos.

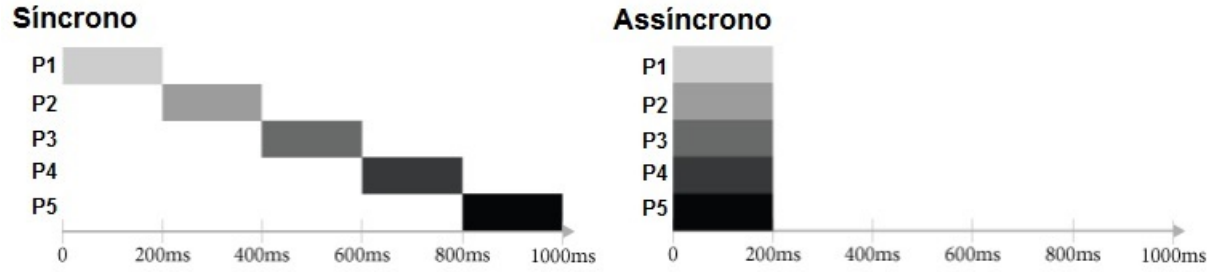
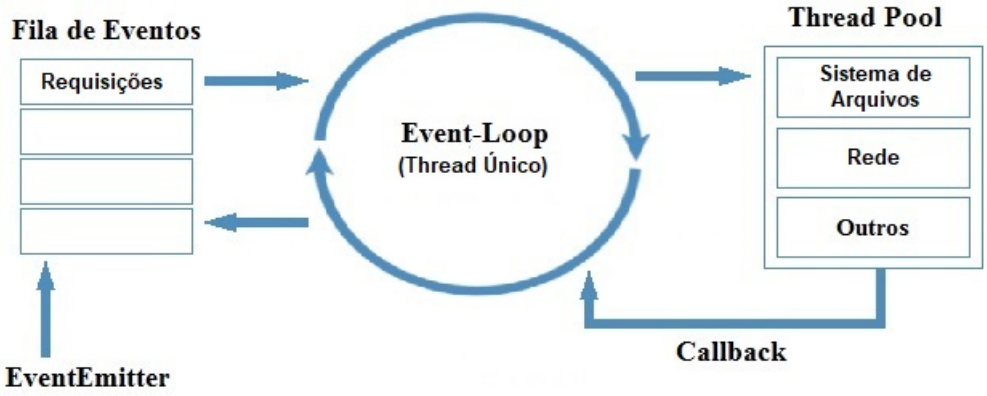


Figura 7 – Event-Loop no Node.js.



2.3.2 Express

Apesar da simplicidade do Node.js, conforme são implementados novas funções a aplicação e a complexidade do sistema aumenta, acaba se tornando trabalhoso e inviável a criação de rotas (*Routes*) para cada uma delas. Foi dessa premissa que surgiu o Express (NODE.JS FOUNDATION, 2018b), um framework para servidores de aplicação web baseados em Node.js que fornece uma série de recursos que simplificam o uso de roteamento e operações HTTP, evitando que estes tipos de serviços precisem ser manualmente implementados no servidor Node.js. Ele age no meio de uma solicitação bruta do usuário e uma rota final desejada.

As principais características do framework são: MVR (*Model-View-Routes*), MVC (*Model-View-Controller*), roteamento de URLs via *callbacks*, *middleware*, interface RESTFul e integração com SQL e NoSQL (PEREIRA, 2013). Desta forma, o Express se encarrega de realizar o roteamento entre as solicitações do usuário e o destino específico na *View*. Podemos dizer que ele é o responsável pela integração do servidor com o AngularJS no lado cliente.

A definição básica de uma rota nesta ferramenta segue a estrutura app.METHOD (PATH, HANDLER) (NODE.JS FOUNDATION, 2018b), sendo o app uma instância do Express, o METHOD um- método de requisição HTTP (get, post, put ou delete), o PATH

um caminho URL (ou parte dele) no servidor e o HANDLER uma ou mais funções de manipulação que são executadas caso a rota seja correspondida. A Listagem 2 ilustra a definição de uma rota em Express que responde o popular *Hello Word* no index da aplicação:

```
1 //app.METHOD(PATH, HANDLER)
2 app.get('/', function (req, res) {
3     res.send('Hello World!'); }
```

Listagem 2 – Definição de uma rota com Express.

Adicionalmente, ao instalarmos o módulo Express através do gerenciador npm e criarmos um projeto através do comando terminal "express novo_projeto -e ejs", o framework já cria toda a estrutura de diretórios que serão utilizadas para o desenvolvimento de toda a aplicação, como o "public/" (repositório de arquivos estáticos como imagens, javascript e css), o "routes/" (repositório das rotas definidas) e o "views/" (diretório das *views* renderizadas pelas rotas), incluindo o arquivo "app.js" que inicializa o servidor e o arquivo "package.json" que inclui e gerencia todos os módulos já instalados no projeto. Vale ressaltar que essa distribuição de arquivos pode ser alterada conforme necessidades da aplicação.

2.3.3 Mongoose

O Mongoose é um módulo também instalado via npm, responsável pela integração do servidor com o banco de dados. Apesar de seu uso não ser obrigatório, esta ferramenta facilita a estruturação dos dados na aplicação através de modelos chamados de *Schemas* antes de armazena-los ao banco de dados, sendo que cada esquema mapeia uma coleção dentro do MongoDB e representa a forma dos documentos dessa coleção.

O Mongoose é bastante utilizado para a implementação de estruturas de dados para o MongoDB (banco de dados utilizado pela pilha MEAN), e isso se deve ao fato de que seus *Schemas* se baseiam em um conjunto de chaves do tipo atributo : valor, que é o mesmo formato utilizado em documentos em formato JSON como é o caso dos documentos gerados pelo MongoDB, permitindo uma modelagem de dados transparente e de fácil implementação. Nestas chaves, os valores consistem de variáveis *SchemaType* que aceitam os tipos String, Number, Date, Buffer, Boolean, Mixed, ObjectId e Array. Um exemplo de um modelo Mongoose utilizado no desenvolvimento deste trabalho pode ser visto na Listagem 3, onde é criado um novo *Schema* para as leituras chamado "leituraSchema" que receberá as informações obtidas pelo servidor dos módulos em campo e os atribuirá a atributos relacionados a cada sensor e por fim criando um *Model* a partir do esquema criado com o qual o servidor poderá trabalhar e armazenar documentos no banco de dados, com a ressalva de que a modelagem de dados será abordada mais adiante na Seção 2.4.1.

```
1 var mongoose = require('mongoose');
```

```
2 var Schema = mongoose.Schema;
3
4 var leituraSchema = new Schema({     //Novo Mongoose Schema
5     id_leitura: Number,      //Atributo: SchemaType
6     id_sensor:  Number,
7     horario_leitura: {  type: Date, default: Date.now },
8     ambiente:{  temperatura: Number,
9                 umidade: Number,
10                luminosidade: Number
11                folha: Number }
12    solo:    {   temperatura: Number,
13                umidade: Number }
14 });
15
16 //Cria um Modelo baseado em um Schema
17 var Leitura = mongoose.model('Leitura', leituraSchema);      //mongoose.
    model(nomeModelo, nomeSchema)
```

Listagem 3 – Criação de um Modelo de Dados com o Mongoose.

Desta forma, sempre que for necessário mudar a estrutura de novos documentos para o banco de dados (como adicionar ou remover um novo tipo de sensor na leitura), basta alterar o Mongoose Schema direto na aplicação com a inclusão ou remoção de chaves. Com isso as novas leituras armazenadas incluirão os novos campos sem que seja necessário realizar alterações no banco de dados.

2.4 Banco de Dados

Devido a característica de constate monitoramento do campo na agricultura de precisão, são geradas novas leituras ao longo do tempo o que produz um alto volume de dados. Diante disto é necessário que esses dados sejam armazenados de forma eficiente e persistente para gerenciamento e recuperação dessas informações, e por esta razão são integrados bancos de dados a esses tipos de sistemas.

Para esta função foi adotado o MongoDB, uma popular solução utilizada em conjunto ao Node.js e parte da pilha MEAN, sendo um banco de dados NoSQL de código aberto baseado em documentos, que oferece esquemas dinâmicos e flexíveis, trabalhando nativamente com documentos em formato semelhante ao JSON, que se torna ideal para ambientes JavaScript como é o caso do Node.js. Devido a sua flexibilidade, fica mais fácil lidar com estruturas de dados que precisam ser modificadas com o tempo (uma necessidade diante de aplicações IoT), adaptando o *Schema* de documentos com novos campos sem ter que lidar com a reestruturação de todo o banco de dados (MONGODB, INC, 2018).

O JSON é um formato de armazenamento e envio de dados que surgiu como al-

ternativa ao XML (*eXtensible Markup Language*), mas se tornou popular devido a sua semântica aproximar a compreensão humana com a da máquina. A estrutura de um documento deste tipo é baseada em um conjunto de pares atributo : valor (JSON.ORG, 2018).

Além disso, a plataforma MongoDB disponibiliza multiplas *Engines* de Armazenamento, ditas *Storage Engines*. Por padrão, nas versões mais atuais do MongoDB, é utilizado a *storage engine WiredTiger* que permite o armazenamento e acesso de arquivos que não excedam o tamanho limite de 16MB do documento BSON (*Binary JSON*).

Uma alternativa ao WiredTiger é o GridFS. Esta engine difere das demais, pois ao invés de armazenar os dados em um único documento, como é comum no Mongo, ela divide o arquivo em partes chamadas de *chunks* de um mesmo tamanho (255kB, por padrão), e armazena cada chunk em um documento separado, onde somente o último varia com o tamanho remanescente. Com isso, no banco de dados, é preciso utilizar no mínimo duas coleções, uma para armazenar os documentos com os *chunks* e outra para armazenar o metadata. Essa é a solução adequada para armazenar arquivos que ultrapassem os 16MB limites das aplicações padrões.

Antes de prosseguir, é importante destacar as diferenças entre um banco de dados SQL relacional (RDBMS – *Relational Database Management System*) e um banco de dados NoSQL orientado a documentos, como é o caso do MongoDB. Essas diferenças são definidas na Tabela 1.

2.4.1 Modelo de Dados Documental

A principal diferença entre os banco de dados SQL relacional e o MongoDB está na modelagem e manipulação dos dados armazenados. Na modelagem relacional, toda a estrutura de dados do sistema é definida e distribuída em uma série de tabelas que se relacionam entre as outras através de chaves primárias. Já no MongoDB, o interesse é agrupar as informações de interesse em um único documento contendo todos os atributos.

Dentro deste escopo, o Mongo ainda permite armazenar documentos com estruturas diferentes em uma mesma coleção. Assim, é possível, por exemplo, termos em uma coleção de pessoas uma instância de objeto que possua o atributo “telefone” e uma segunda instância que não possua, como observado na Listagem 4.

```
1 db.pessoa =
2 {           // Primeiro documento na colecao Pessoa
3     "_id" : 01,
4     "nome" : "Carol",
5     "sobrenome" : "Magalhaes"
6 }
7 {           // Segundo documento na colecao Pessoa
8     "_id" : 02,
9     "nome" : "David",
```

```
10      "sobrenome" : "Fontes"
11  }
12  {           // Terceiro documento na colecao Pessoa
13      "_id" : 03,
14      "nome" : "Rebeca",
15      "sobrenome" : "Tavares",
16      "telefone": "(63)9999-3333"
17  }
```

Listagem 4 – Criação de um Modelo de Dados com o Mongoose.

Tabela 1 – Relação de Terminologias entre RDBMS e MongoDB

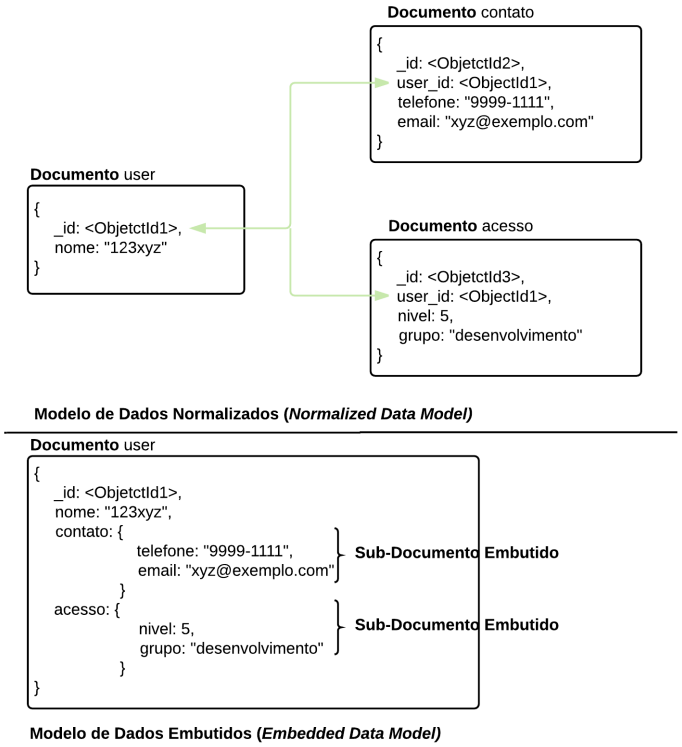
RDBM	MongoDB	Relação
Tabela (*Table*)	Coleção (*Collection*)	Nos RDBMs, são criadas tabelas que possuem Colunas e Linhas para armazenamento dos dados, enquanto que no Mongo essa estrutura é dada como Coleções contendo Documentos que possuem uma série de Campos que armazenam pares do tipo chave-valor.
Linha (*Row*)	Documento (*Document*)	Em RDBMs, uma linha representa o registro de um item na estrutura de dados da tabela, enquanto que no MongoDB, cada instância de um objeto é armazenado em forma de Documentos separados.
Coluna (*Column*)	Campo (*Field*)	Em RDBMs, uma coluna representa um atributo do objeto com um tipo de valor específico. Da mesma forma, no lado do Mongo esses atributos são dados como Campos.
Inclusões (*Joins*)	Documentos Embutidos (*Embedded Documents*)	Em RDBMs, são usados joins entre diversas tabelas para acessar todos os dados de uma estrutura relacional. Já no Mongo, sua modelagem permite a criação de documentos dentro de documentos, permitindo que todos os dados sejam incluídos em um único objeto e retornados por uma única query.

Isso remove a necessidade de preenchimento de campos com valores NULL, como se faz em tabelas SQL, reduzindo a sobrecarga no banco e agilizando consultas query uma vez que, no exemplo acima, somente um documento seria retornado em uma busca com filtro baseado no atributo “telefone”. De forma semelhante, outro recurso que otimiza o desempenho de funções query no MongoDB é o fato de todas as informações serem acessadas a partir do retorno de um único objeto, sem a necessidade de execução de *joins* como acontece em bancos de dados relacionais.

No MongoDB, temos basicamente dois modelos de dados: Modelo de Dados Normalizados (*Normalized Data Models*) e Modelo de Dados Embutidos (*Embedded Data Models*). O primeiro se aproxima mais de quando pensamos numa modelagem relacional,

onde são criados um documento pra cada classe e o relacionamento entre eles se dá pela inclusão do ObjectID nas classes relacionais (como ocorre com as chaves primárias no SQL). O ObjectID é um campo único e criado automaticamente pelo MongoDB ao gravar um documento na base dados, assim, cada documento salvo possui sua própria identidade. Já no Modelo de Dados Embutidos, todos os atributos são representados por um único documento, de forma simplificada e direta tendo implícito o relacionamento, já que a arquitetura do Mongo permite a inclusão de documentos dentro de outros documentos. A Figura 8 mostra a equivalência entre os dois modelos de dados.

Figura 8 – Modelo de Dados Normalizados (acima) x Modelo de Dados Embutidos (abaixo) (MONGODB, INC, 2018).

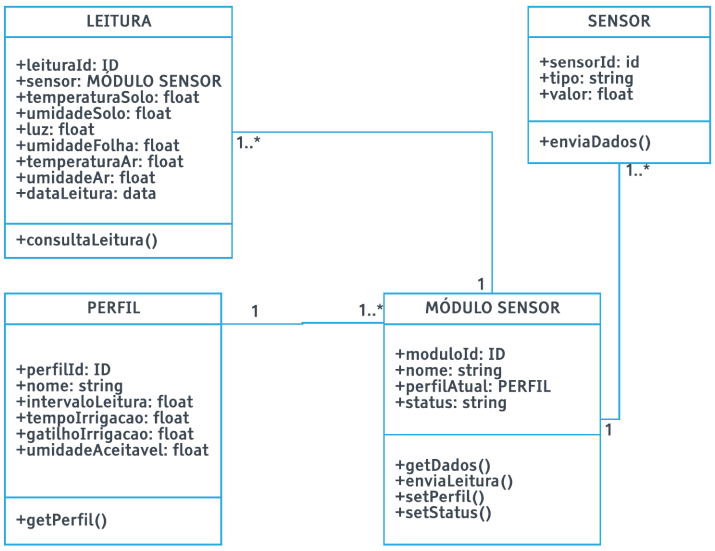


Definimos três coleções de documentos para representar os dados do projeto, divididos em Leituras, Perfis e Módulos. Na Figura 9 é visto o Diagrama de Classes do sistema de irrigação automática proposto, e através dela é possível uma melhor compreensão sobre a relação dessas entidades no projeto.

A Listagem 5 representa o modelo de dados criado para armazenar os dados das leituras no banco de dados, definindo o padrão em que as informações dos sensores são salvas no servidor. A Listagem 6 representa o modelo de dados criado para armazenar os perfis de cultivo cadastrados pelo usuário, servindo como um padrão de configuração para variáveis de rotina dos módulos sensores em campo. Por fim a Listagem 7 representa o modelo de dados dos módulos cadastrados, dando ao servidor referência quanto a utilização dos módulos em campo.

Vale ressaltar que esses são modelos genéricos concebidos para o escopo da solução

Figura 9 – Diagrama de Classes do Sistema de Irrigação Automática.



almejada, e seus atributos podem facilmente ser modificados e novos atributos inseridos conforme necessidades específicas da plantação e disponibilidade de novos tipos de sensores.

```
//Documento Leitura
{
    _id: <ObjectID> //Atribuido automaticamente e referencia o Documento
    idLeitura: <Numero> //Id incremental do numero da leitura
    sensor: <String>, //Nome do Modulo Sensor responsavel pela leitura
    temperaturaSolo: <Numero>, //Valor ambiente na regiao do sensor
    umidadeSolo: <Numero>, //Valor ambiente na regiao do sensor
    luz: <Numero>, //Valor ambiente na regiao do sensor
    umidadeFolha: <Numero>, //Valor ambiente na regiao do sensor
    temperaturaAr: <Numero>, //Valor ambiente na regiao do sensor
    umidadeAr: <Numero>, //Valor ambiente na regiao do sensor
    dataLeitura: <Data> //Data de recebimento da leitura
}
```

Listagem 5 – Modelo de Dados para Coleção de Leituras.

```
//Documento Perfil
{
    _id: <ObjectID> //Atribuido automaticamente e referencia o Documento
    idPerfil: <Numero> //Id incremental do numero do Perfil
    nome: <String>, //Nome do Perfil de plantio
    intervaloLeitura: <Numero>, //Valor em minutos do intervalo de
        requisicao de leituras
    intervaloIrrigacao: <Numero>, //Valor em minutos em que e verificada
        a necessidade de irrigacao
    gatilhoIrrigacao: <Numero>, //Valor de umidade do solo que ativa
```

```
        irrigacao
9       umidadeAceitavel: <Numero> //Valor de umidade do solo que desativa
        irrigacao
10  }
```

Listagem 6 – Modelo de Dados para Coleção de Perfis.

```
1  //Documento Perfil
2  {
3       _id: <ObjectID> //Atribuido automaticamente e referencia o Documento
4       nome: <String>, //Nome do Modulo Sensor
5       perfilAtual: <String>, //Referencia para o perfil utilizado pelo
        modulo
6       localizacao: <String>, //Referencia de localizacao do modulo no
        campo (zonas ou coordenadas, por exemplo)
7       statusAtivo: <String> //Indica se o modulo esta ativo ou inativo
8  }
```

Listagem 7 – Modelo de Dados para Coleção de Perfis.

2.5 Aplicação Web

Como módulo final do projeto, é necessária uma ferramenta que execute no lado do cliente, ou seja, que permita ao usuário utilizar os recursos do sistema. Para isto, será desenvolvida uma aplicação web acessível por navegadores da Internet. Esta aplicação tem como principais funções: exibir as informações armazenadas no banco de dados em forma de tabelas, exibir uma representação gráfica dos dados e permitir ao usuário gerenciar e interagir com os módulos em campo.

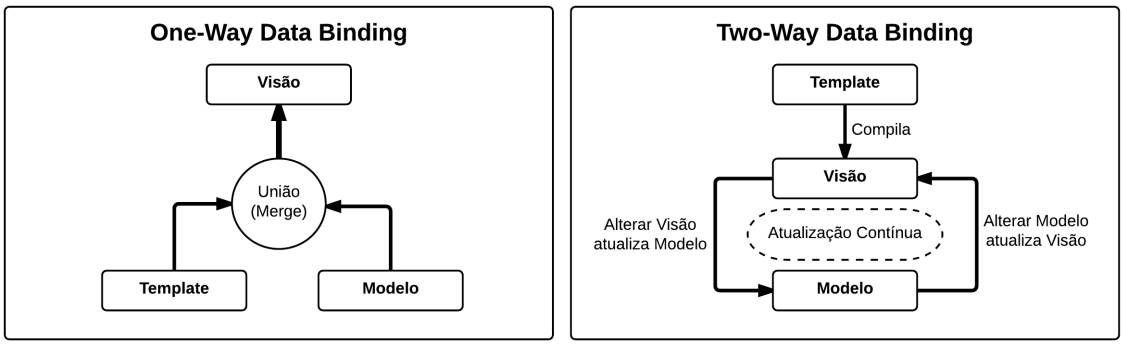
2.5.1 AngularJS

Para o desenvolvimento da aplicação web, foi utilizado o AngularJS, um framework mantido pela Google para o desenvolvimento de aplicações em sua arquitetura MVC, estendendo o vocabulário HTML e permitindo a sincronização automática de *models* e *views* através do seu *Two-Way Data Binding*. Seu uso é ideal para o desenvolvimento de SPAs (*Single Page Applications*, ou Aplicações de Página Única) e funciona com qualquer tipo de HTML, como o HTML 5. Dentre suas principais vantagens, está o fato de que o framework remove, através de suas diretivas, a necessidade de manipularmos o DOM em baixo nível manualmente, não possui requisitos de *backend* (podendo ser utilizada a linguagem de preferência) e o desenvolvimento no AngularJS se dá em menos linhas de código do que uma solução JavaScript pura, reduzindo o trabalho dos programadores (GOOGLE, INC, 2018a).

O *Two-Way Data Binding* no AngularJS é a sincronização automática de dados entre os componentes de modelo e visão, implementando uma vinculação de dados. Desta

forma, a visão é uma projeção do modelo em todos os momentos: quando o modelo muda, a visão reflete a mudança e vice-versa, como pode ser visto na Figura 10, evitando que o desenvolvedor tenha que se preocupar em escrever o código que faça essa sincronização.

Figura 10 – Two-Way Data Binding no AngularJS (GOOGLE, INC, 2018a).



Dando suporte ao padrão MVC, o AngularJS possui alguns conceitos importantes: o $scope, o ng-model e o ng-controller. O *scope* é um objeto no AngularJS que armazena os dados da aplicação e, por isso, ele é responsável pela ligação entre o Controller e a View. Como os dados são armazenados em sua estrutura, o *scope* é utilizado para auxiliar o *data binding* realizado em expressões do AngularJS. A implementação do *Two-Way Data Binding* se dá através do "ng-model", a diretiva responsável por vincular valores de controle HTML (como inputs, selects, textarea, etc) aos dados da aplicação, atualizando os valores no Modelo (*scope*). Já os *Controllers*, que como vimos são responsáveis por intermediar as camadas de modelo e visão, são declarados na *view* e recebem um $scope por injeção de dependência. Em outras palavras, o controlador fará com que as variáveis do escopo sejam levadas a *view* e até mesmo como elas se comportarão (com o uso de funções dentro do próprio escopo, por exemplo).

É possível ver na Listagem 8 uma construção simples que mostra a relação entre modelo, controlador e visão no AngularJS. No exemplo, as variáveis do escopo são vistas pelo usuário na div através do controlador "meuControlador" declarado na *view* e, caso o usuário altere qualquer dos campos input, o *Two-Way Data Binding* fará a atualização dos valores no *scope* automaticamente, criando uma sincronização entre a camada de Modelo e Visão.

```
1 <div ng-app="exemploApp" ng-controller="meuControlador">
2
3 Nome: <input type="text" ng-model="nome"><br>
4 Sobrenome: <input type="text" ng-model="sobrenome"><br>
5 <br>
6 Nome Completo: {{nomeCompleto()}} //Expressao AngularJS
7 </div>
8
9 <script>
```

```
10 var app = angular.module('exemploApp', []);
11 app.controller('meuControlador', function($scope) {
12     $scope.nome = "Lucas";
13     $scope.sobrenome = "Beraldo";
14     $scope.nomeCompleto = function() {
15         return $scope.nome + " " + $scope.sobrenome;
16     };
17 });
18 </script>
```

Listagem 8 – Construção do Two-Way Data Binding com Angular.

O Angular também permite integração com Bootstrap, possibilitando o uso de recursos CSS junto à aplicação para a exibição de um conteúdo limpo (BOOTSRAP, 2018).

2.5.2 *Website Design*

A aplicação web foi desenvolvida da forma mais simples possível, priorizando o acesso aos recursos do sistema e suas funcionalidades para a solução proposta acima da estética, não sendo apresentada como um produto final. Assim, partimos de uma página inicial com uma espécie de menu com as principais funcionalidades do servidor: Consultar Leituras, Gerenciar Perfis e Gerenciar Módulos.

Para consulta de leituras, a aplicação oferece uma página que lista em tabela todas as leituras salvas no banco de dados do servidor, oferecendo a opção de acessar cada uma dessas leituras para visualizar suas informações detalhadas e também a opção de gerar gráficos baseados nos dados registrados, como pode ser visto na Figura 11.

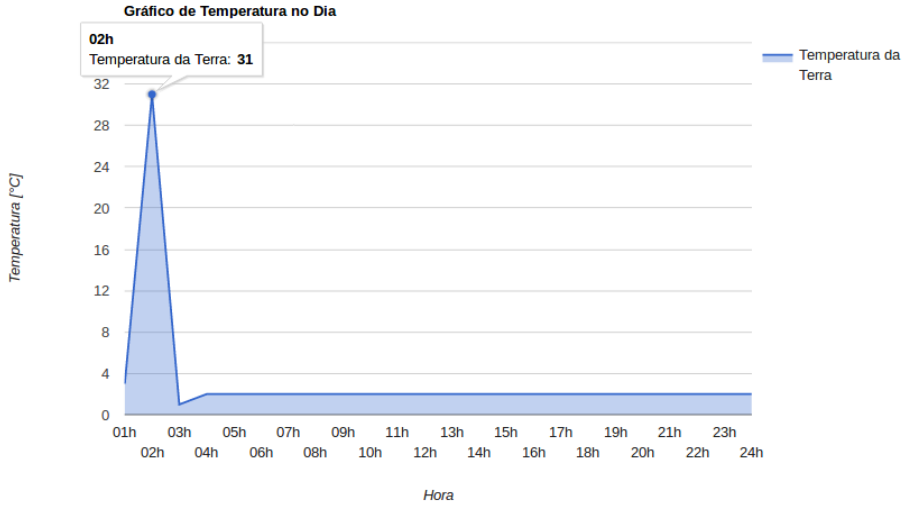
Atendendo a necessidade de exibir graficamente as informações das leituras, implementamos na aplicação um módulo do Google Charts chamado Angular Google Charts, disponível para instalação pelo npm do Node.js, facilitando a visualização da informação através da geração de gráficos com os dados do servidor. Com o Google Charts, é possível criar gráficos interativos de diferentes tipos como gráficos de área, pontos, barras entre outros para navegadores e dispositivos móveis (GOOGLE, INC, 2018b).

Para esta solução preferimos usar um gráfico de área dinâmico que apresente os valores das leituras baseadas no tipo de sensor. Há inúmeras possibilidades de representar os dados do servidor, então como um exemplo genérico decidimos exibir a média das leituras ao longo do dia em uma determinada data escolhida pelo usuário, como visto na Figura 12.

Para o gerenciamento de perfis, temos uma página semelhante a de leituras, onde são listados os perfis cadastrados no sistema com link para suas informações, além da opção de cadastrar um novo perfil (Figura 13). Assim, conforme necessidade, o usuário pode adicionar diversos perfis de diferentes cultivos e utilizá-los posteriormente na configuração

Figura 11 – Tela de Consulta de Leituras na Aplicação Web.

Figura 12 – Exemplo de geração de gráfico com Google Chart.



de novos módulos sensores integrados ao sistema.

Para o gerenciamento de módulos, também criamos uma página de listagem em tabela dos módulos cadastrados e a opção de cadastro de novo módulo permitindo que, sempre que um novo módulo sensor seja inserido na rede de sensores, ele seja adicionado ao sistema com um perfil de cultivo. Desta forma, o usuário tem acesso às informações de cada módulo cadastrado como visto na Figura 14, podendo gerenciar o seu status de atuação e o perfil em vigor. Ao acessar o link de um módulo específico, o usuário é capaz de interagir com o mesmo remotamente, podendo solicitar uma leitura deste sensor a qualquer momento, enviar uma requisição para que o módulo ligue/desligue (baseado no

Figura 13 – Tela de Cadastro de Perfis na Aplicação Web.

seu status) e o mais importante: sempre que o perfil do módulo for alterado, é possível reconfigurar as variáveis de operação do módulo sem a necessidade de reprogramá-lo manualmente ao publicar as informações do novo perfil para o tópico que o dispositivo é assinante através do protocolo MQTT.

Figura 14 – Tela de Gerenciamento de Módulo por ID na Aplicação Web.

Resumindo, a aplicação desenvolvida permite ao cliente acesso e visualização dos dados do servidor e também interagir com os dispositivos em campo. No entanto a conexão com o *broker* do CloudMQTT existe apenas no lado servidor, sendo necessário transmitir ao servidor as requisições feitas pelo usuário para que então elas possam ser publicadas em seus respectivos tópicos. Para esta solução decidimos implementar no servidor a biblioteca do Socket.IO.

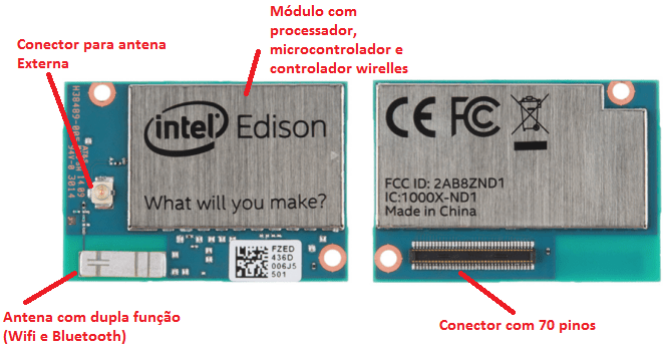
O Socket.IO habilita uma comunicação bidirecional baseada em eventos em tempo real que funciona em toda plataforma, navegador ou dispositivo, com foco em confiabilidade e velocidade, sendo muito utilizado em aplicações de troca de mensagem instantânea. Ele é composto por duas partes: um servidor que integra o servidor HTTP Node.js e uma biblioteca cliente que carrega no lado do navegador (SOCKET.IO, 2018). Essa biblioteca também pode ser obtida através do Gerenciador de Pacotes do Node.js, o NPM.

2.6 System on Chip: Intel Edison

Um dos objetivos desse projeto é o desenvolvimento de um sistema de baixo custo e consumo de energia diferindo de servidores mais robustos que demandam grande inves-

timento em *hardware*, e devido a isso tomamos a decisão de embarcar o sistema em um dispositivo SoC. Em sistemas embarcados, os dispositivos são utilizados para executar tarefas específicas e acabam reduzindo o tamanho, custo e recursos computacionais do projeto.

Figura 15 – Imagem do módulo Intel Edison e seus componentes.



Um sistema SoC chega a ser semelhante a microcontroladores, no entanto, distinguem-se no fato de ser um sistema completo em um único *die* (pequeno bloco de material semicondutor no qual o circuito funcional é fabricado) e possuírem processadores mais potentes (capazes de executar programas e sistemas operacionais).

O servidor atuará diretamente no dispositivo, e para isto foi escolhido o Intel Edison cujo sistema operacional Yocto Linux oferece suporte nativo a Node.js, que já vem pré instalado neste modelo. Assim, será possível aproveitar diretamente o desenvolvimento do sistema feito. Esse módulo é um microchip projetado para a construção de produtos Internet das Coisas (IoT), contendo uma unidade de processamento dual-core de alta velocidade, Wi-Fi integrado, Bluetooth de baixa energia, armazenamento e memória (INTEL, 2018a).

2.6.1 Hardware

A estrutura geral do módulo Intel Edison pode ser vista na Figura 16, e seus principais componentes de hardware listados na Tabela 2.

O módulo consiste de um processador Intel Atom que opera a uma velocidade de clock de 500 MHz e 4 GB de memória flash gerenciada. Como é um dispositivo IoT, ele possui para conectividade Wi-Fi e Bluetooth de baixa energia um chip Broadcom BCM43340 que suporta dual-band de 2,4 GHz e 5 GHz no padrão IEEE 802.11 a/b/g/n, *Wi-Fi Protected Access* (WPA) E WPA2 para encriptação e autenticação. Com essa opção de conectividade, ele facilita a conexão de dispositivos com módulos Edison à infraestrutura Wi-Fi existente de forma padronizada. Além disso, o módulo inclui um conector para antena externa que pode ser utilizado para ampliar o alcance do sinal.

O Bluetooth de baixa energia permite que os dispositivos Edison se conectem a outros dispositivos Bluetooth de baixa energia, como smartphones configurados para

atuarem como um gateway para se conectar à Internet. (INTEL, 2018b)

Figura 16 – Diagrama de Blocos do Módulo Intel Edison (INTEL, 2018b).

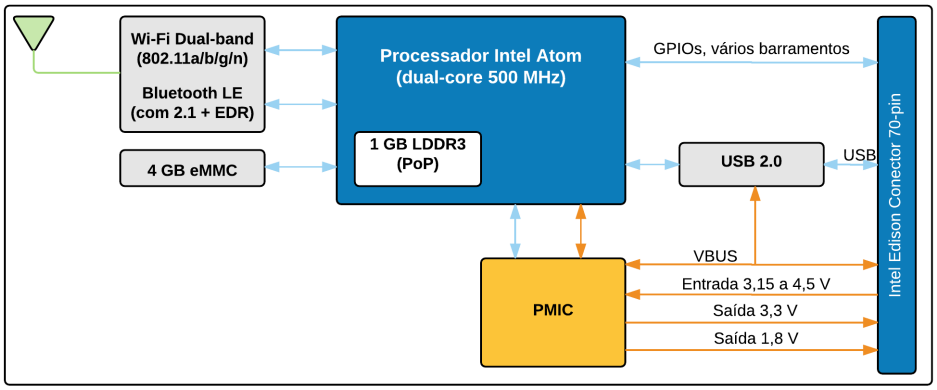


Tabela 2 – Componentes de Hardware do Intel Edison.

Componente	Descrição
Processador	Intel SoC de 22 nm que inclui um dual-core/dual-threaded Intel Atom CPU a 500 MHz e um microcontrolador 32-bit Intel Quark a 100 MHz
RAM	Mémoria de 1 GB LPDDR3 PoP
Armazenamento	4 GB eMMC
Energia	TI SNB9024 PMIC
Wireless	Dual-band (2.4 e 5 GHz) IEEE 802.11a/b/g/n
Bluetooth	BT 4.0 + 2.1 EDR
Antena	Antena onboard Dual-band ou u.FL para antena externa
Conector	70-pin Hirose DF40 Series
Tamanho	35,5 x 25,0 x 3,9 mm
Input de Energia	3,15 a 4,5 V
USB 2.0	1 controlador OTG
Clocks	19,2 MHz, 32 kHz

3 METODOLOGIA

O projeto se iniciou com a preparação do ambiente de trabalho, e para isso contou-se com a criação de uma máquina virtual no software *VMware Workstation* baseada no sistema operacional Linux Ubuntu 14.10, sendo esta executada em um notebook Dell Inspiron 5548 com processador Intel(R) Core(TM) i7-5500U de 2.40GHz e 16GB de Memória RAM DDR3.

O próximo passo no setup do ambiente de desenvolvimento foi a instalação do Node.js e o MongoDB na máquina, sendo o primeiro concluído com facilidade através do *download* dos instaladores disponíveis no site da própria organização para diferentes plataformas. No momento em que foi iniciado o desenvolvimento deste projeto, o Node.js estava em sua versão 4.2.6. Já para o MongoDB foi utilizado o MongoDB Community Edition (em sua versão 3.4.7, disponível no período de desenvolvimento) obtido através do apt-get do Linux, tendo seu processo de instalação detalhado em sua documentação oficial para cada tipo de sistema operacional. A Listagem 9 mostra como verificar se ambos os softwares estão devidamente instalados e executando na máquina.

```
1 // Arquivo helloworld.js criado
2 console.log("Hello, World!")
3
4 //Terminal do Linux
5 //Comando utilizado para executar arquivos Javascript do Node.js
6 $ node helloworld.js
7 //Resultado no console apos execucao do arquivo
8 Hello, World!
9
10 //Comando para executar o servico do MongoDB instalado
11 $ sudo service mongod start
12
13 //Se inicado corretamente, o banco de dados esta pronto para uso e seu
      console pode ser acessado com o seguinte comando
14 $ mongo
```

Listagem 9 – Execução do Node.js e MongoDB na máquina.

Com o Node.js instalado a máquina virtual agora tem acesso ao NPM, o Gerenciador de Pacotes do Node, e em seu acervo de pacotes temos o necessário para obter os demais componentes requeridos no projeto: Express, MongoDB Node.js (um driver que fornece interação com o serviços do MongoDB), Mongoose, Angular, MQTT.js (biblioteca do protocolo MQTT disponível para Node.js) e Angular Google Chart. A Listagem 10 mostra o processo de instalação dos pacotes e a criação do diretório do projeto com o Express. Como o desenvolvimento do projeto foi iniciado antes da escrita desta monografia

também são listadas as versões disponíveis na data de implementação do trabalho.

```
1 //NOTA: O comando --save serve para adicionar o pacote na lista de
      dependencias do projeto atraves do arquivo package.json
2
3 // Instala o pacote do ExpressJS (disponivel em sua versao 4.15.2) e seu
      gerador de projetos
4 $ npm install express-generator -g
5
6 // Cria e acessa um diretorio para os arquivos do projeto
7 $ mkdir projeto_dir
8 $ cd projeto_dir
9
10 // Cria uma estrutura basica de aplicativo no diretorio atual
11 $ express [options] nome_projeto
12
13 // Instala o driver do MongoDB para Node.js (disponivel em sua versao
      2.2.31)
14 $ npm install mongodb --save
15
16 // Instala a biblioteca Moongose (disponivel em sua versao 4.11.7)
17 $ npm install mongoose --save
18
19 // Instala o biblioteca AngularJS (disponivel em sua versao 1.3.10)
20 $ npm install angular@1.3.10
21
22 // Instala o biblioteca MQTT.js (disponivel em sua versao 2.13.0)
23 $ npm install mqtt --save
24
25 // Instala o biblioteca Socket.io (disponivel em sua versao 2.1.1)
26 $ npm install socket.io --save
```

Listagem 10 – Instalação de Pacotes via NPM.

Uma vez instaladas, essas bibliotecas podem ser acessadas no principal arquivo Javascript do servidor Node.js através do método *require*. Este método pode ser utilizado para carregar tanto bibliotecas quanto outros arquivos Javascript em diferentes diretórios do projeto em variáveis para serem utilizadas pelo servidor, como visto na Listagem 11. Nela também é exibido como são estabelecidas as conexões com os principais serviços utilizados pelo servidor.

Com essas ferramentas foi possível desenvolver o servidor propriamente dito, como também a aplicação do usuário. Devido a sua extensão, não convém entrar em detalhes sobre o desenvolvimento de cada função mas, resumidamente, foram implementados: os métodos de envio/recebimento de solicitações entre cliente/servidor com a conexão Socket.io e entre servidor/sensores com a conexão MQTT; os esquema Mongoose para os modelos de Leituras, Perfis e Módulos no diretório '*models*' do projeto; as páginas HTML

da nossa aplicação web no diretório '*public/views*'; as rotas responsáveis por atender as requisições HTTP cliente/servidor no diretório '*routes*' e por fim os controladores no diretório '*public/javascripts*', responsáveis por servir as páginas da aplicação web com os dados persistidos no banco de dados.

```
1 //Arquivo main.js
2
3 //Carrega os pacotes das dependencias atraves do metodo require
4 var express = require('express');
5 var mqtt = require("mqtt");
6 var mongoose = require('mongoose');
7 var http = require('http');
8 var SocketIO = require('socket.io');
9
10 //Inicia servidor HTTP do Node
11 var server = http.createServer(app);
12
13 // Conecta ao \textit{broker} do CloudMQTT
14 var clientMQTT  = mqtt.connect('mqtt://m13.cloudmqtt.com:PORT',{
15     username: 'user',
16     password: 'password'
17 });
18
19 //Inicia servidor do Socket.io montado no servidor Node.js
20 var io = SocketIO(server);
21
22 //Conecta ao servico do banco de dados do MongoDB atraves do mongoose
23 var mongodb = mongoose.connect('mongodb://localhost/servidor', {
24     useMongoClient: true,
25 });
```

Listagem 11 – Estabelecendo conexão com os serviços do Sistema.

Estabelecidas as conexões, foi possível executar a aplicação para testar o seu funcionamento e o fluxo de informações. Uma das funções mais importantes do servidor é o recebimento e armazenamento dos dados de leitura dos sensores em campo, e para isso foi utilizada a biblioteca MQTT para Node.js. É possível ver na Listagem 12 o trecho de código que gerencia o recebimento e persistência das informações no banco de dados do servidor.

```
1 //Arquivo main.js
2
3 //Carrega o modelo de dados Leitura e cria um objeto global do mesmo
      tipo
4 var Leitura = mongoose.model('Leitura');
5 var leitura = new Leitura();
6
```

```
7 // Estabelece conexao como visto anteriormente
8 //(...)
9
10 clientMQTT.on("connect", function () {
11     //Assina topico 'iam' e todos os seus subtopicos
12     clientMQTT.subscribe("iam/#");
13     console.log("Conectou no CloudMQTT");
14 });
15
16 clientMQTT.on('message', function (topico, mensagem) {
17     if(topico == 'iam/modulos/microclima/[nome_modulo]/leitura/[
           tipo_leitura]')
18     {
19         var num = parseFloat(mensagem);
20         //Atribui o valor enviado pela mensagem no atributo
               correspondente do objeto leitura
21         //Ex. Se [tipo_leitura] for data, atributo sera leitura.
               dataLeitura
22         leitura.[atributo_Leitura] = num;
23         if(verificaLeitura())
24         {
25             //Atribui nome do sensor que enviou a leitura
26             leitura.sensor = [nome_modulo];
27             //Cria um novo documento na colecao 'Leituras' do MongoDB
                   com o objeto 'leitura'
28             //A funcao save e gerenciada pelo Mongoose
29             leitura.save(function(err) {
30                     if(err)
31                         return next(err);
32                 });
33             //Reseta o objeto 'leitura' para receber a proxima leitura
34             leitura = new Leitura();
35         }
36     }
37     else if{[topico == outro_topico]} //Repete o processo para cada
           topico de leitura do broker
38     {
39         //...
40     }
41 }
```

Listagem 12 – Recebendo dados de sensores do CloudMQTT.

Uma vez conectado ao *broker* do CloudMQTT, o servidor realiza a assinatura do tópico raiz do projeto, o "IAM", sinalizando que deseja receber as mensagens enviadas nesse tópico e seus subtópicos, assim, sempre que um módulo sensor finaliza a coleta de dados de uma leitura ele publica essas informações através de uma mensagem nos tópicos

do MQTT correspondentes e esses mensagens são repassados pelo *broker* ao servidor na conexão clientMQTT criada. Com elas, o servidor é capaz de identificar o tipo de informação enviada a partir do tópico da mensagem e atribuir o seu valor no atributo de um objeto global do tipo Leitura correspondente. Para cada mensagem que o servidor recebe um novo atributo do objeto 'leitura' é preenchido e então é verificado se algum atributo permanece vazio (sem atribuição): se sim, a função finaliza indicando que a leitura não está completa e o servidor aguarda a mensagem de outros tópicos; se não, a nova leitura está completa sendo criado um novo documento na coleção de Leituras do banco de dados do servidor com as informações de 'leitura' e posteriormente resetando esse objeto para que o mesmo possa receber os dados da próxima rotina de leitura. A relação entre o tópico do MQTT e o valor passado em mensagem pode ser vista na Tabela 3.

Tabela 3 – Relação de tópicos do CloudMQTT e valores recebidos no servidor de aplicação.

Tópico	Mensagem
iam/modulos/microclima/[nomemodulo]/leitura/data	Date instantâneo
iam/modulos/microclima/[nomemodulo]/leitura/luz	valor[Lux]
iam/modulos/microclima/[nomemodulo]/leitura/umidadear	valor[%]
iam/modulos/microclima/[nomemodulo]/leitura/temperaturaarc	valor[∘C]
iam/modulos/microclima/[nomemodulo]/leitura/temperaturaterrac	valor[∘C]
iam/modulos/microclima/[nomemodulo]/leitura/umidadesolo	valor[%]
iam/modulos/microclima/[nomemodulo]/leitura/umidadefolha	valor[%]

O segundo aspecto funcional do servidor é permitir a interação do usuário com os módulos em campo e para isso ambos invertem papéis: o servidor atua como publicador em tópicos específicos do CloudMQTT e o módulo sensor em campo atua como assinante dos mesmos. As requisições implementadas foram do tipo Solicitar Nova Leitura, Ativar/Desativar Módulo e Reconfigurar Perfil: A primeira função, como o nome sugere, permite solicitar uma nova leitura de um módulo específico no campo independente de sua configuração de intervalo de leitura; a segunda permite enviar um comando para ligar ou desligar um módulo baseado no seu status de funcionamento; e a terceira permite, ao alterar o perfil de plantação de um módulo cadastrado no sistema, enviar os parâmetros do novo perfil para que o módulo possa se auto-reprogramar sem a necessidade do usuário intervir manualmente. Como a conexão MQTT só existe no lado do servidor e não no lado do cliente, foi preciso uma forma de transmitir as requisições do usuário para o servidor de forma que esse pudesse repassar a solicitação pro *broker* online, sendo decidido para isso a inclusão do Socket.IO.

Para testar essas funções, foi realizado o acesso da aplicação web em um navegador. A aplicação desenvolvida fornece uma SPA (*Single Page Aplication*) dividida em três

modalidades: consultar leituras, gerenciar perfis e gerenciar módulos. A página Consultar Leituras exibe uma listagem das leituras salvas no banco de dados e também permite gerar gráficos do Google Charts baseados nessas leituras, como visto adiante no Capítulo 4, na página de Gerenciar Perfis é exibida uma listagem de perfis cadastrados no sistema e a opção de adicionar/editar/remover perfis, mas é na página de Gerenciar Módulos onde se pode interagir com os módulos em campo e realizar as requisições citadas.

Ao acessar a gerência de perfis da aplicação web, é exibido uma lista de módulos cadastrados no sistema além da opção de adicionar/editar/remover módulos. Ao acessar um dos módulos listados é possível ver as informações detalhadas do mesmo (como perfil atual, localização e status de funcionamento) e através de botões de função da página realizar um dos três tipos de requisição citadas para esse módulo. A aplicação angular desenvolvida apresenta um controlador que identifica o tipo de requisição feita e executa uma função específica para a mesma, como pode ser visto na Listagem 13. Para cada função, é enviada uma mensagem via socket pra conexão do servidor, em uma estrutura tópico/mensagem semelhante ao mostrado no MQTT. Assim, ao receber uma mensagem no socket, o servidor consegue identificar o tipo de requisição baseado no tópico da mensagem enviada e realizar uma publicação nos tópicos do CloudMQTT específica para cada caso.

```
1  //Aquivo controllers.js (aplicacao web)
2  app.controller('ModulosCtrl', ['dependencias', function(){
3      const socket = io();
4
5      //Funcoes dos botoes do website
6      $scope.solicitaLeitura = function() {
7          var msg = "solicitaLeitura";
8          //envia mensagem via socket no topico 'solicitaLeitura'
9          socket.emit('solicitaLeitura', msg);
10     };
11
12     $scope.ativaModulo = function() {
13         //Envia a requisicao baseado no status do modulo selecionado
14         if($scope.status == "Ativo")
15         {
16             var msg = "desliga";
17             socket.emit('desligar', msg);
18             //Faz requisicao na rota mudastatus para alterar o status do
                   modulo no banco de dados
19             $http({ url: '/mudastatus', method: 'POST', params: { 'nome'
                  : $scope.nome }, data: { 'ativo': "Inativo" } });
20         }
21         else if($scope.status == "Inativo")
22         {
23             var msg = "liga";
```

```
24              socket.emit('ligar', msg);
25              $http({ url: '/mudastatus', method: 'POST', params: { 'nome'
                    : $scope.nome }, data: { 'ativo': "Ativo" } });
26          }
27      };
28
29      $scope.enviaPerfil = function() {
30          //Verifica na lista de perfis um perfil correspondente ao perfil
                atual do modulo selecionado
31          for(i=0; i<$scope.perfis.length; i++)
32          {
33              if($scope.perfis[i].nome == $scope.nomePerfil)
34              {
35                  //Atribui dados do perfil obtido em variaveis
36                  $scope.nomeSetPerfil = $scope.perfis[i].nome;
37                  $scope.intervaloLeitura = $scope.perfis[i].
                        intervaloLeitura;
38                  $scope.tempoIrrigacao = $scope.perfis[i].tempoIrrigacao;
39                  $scope.gatilhoIrrigacao = $scope.perfis[i].
                        gatilhoIrrigacao;
40                  $scope.umidadeAceitavel = $scope.perfis[i].
                        umidadeAceitavel;
41                  //Envia os valores via socket no topico 'setPerfil'
42                  socket.emit('setPerfil', {nomeSetPerfil: $scope.
                        nomeSetPerfil, intervaloLeitura: $scope.
                        intervaloLeitura, tempoIrrigacao: $scope.
                        tempoIrrigacao, gatilhoIrrigacao: $scope.
                        gatilhoIrrigacao, umidadeAceitavel: $scope.
                        umidadeAceitavel});
43              }
44          }
45      };
46
47  //Arquivo main.js (servidor)
48  var io = SocketIO(server); //conexao socket.io
49
50  io.on('connection', function(socket){
51      //Trata o recebimento de uma mensagem enviada no topico 'desligar'
52      socket.on('desligar', function (msg) {
53          //Publica no topico MQTT a solicitacao de desligar
54          clientMQTT.publish("iam/modulos/microclima/[nome_modulo]/perfil/
                status", "desliga");
55      });
56      socket.on('ligar', function (msg) {
57          clientMQTT.publish("iam/modulos/microclima/[nome_modulo]/perfil/
                status", "liga");
58      });
```

```
59      socket.on('solicitaLeitura', function (msg) {
60          clientMQTT.publish("iam/modulos/microclima/[nome_modulo]/
                solicitacao/leitura", "true");
61      });
62      socket.on('setPerfil', function (data) {
63          //Destrincha os dados da mensagem enviada em variaveis separadas
64          var umidadeAceitavel = data.umidadeAceitavel;
65          var gatilhoIrrigacao = data.gatilhoIrrigacao;
66          var tempoIrrigacao = data.tempoIrrigacao;
67          var intervaloLeitura = data.intervaloLeitura;
68          //Envia cada variavel para o topico MQTT correspondente
69          clientMQTT.publish("iam/modulos/microclima/[nome_modulo]/perfil/
                gatilhofinal", umidadeAceitavel.toString());
70          clientMQTT.publish("iam/modulos/microclima/[nome_modulo]/perfil/
                gatilhoinicial", gatilhoIrrigacao.toString());
71          clientMQTT.publish("iam/modulos/microclima/[nome_modulo]/perfil/
                intervaloirrigacao", (tempoIrrigacao*60).toString());
72          clientMQTT.publish("iam/modulos/microclima/[nome_modulo]/perfil/
                intervaloleituras", (intervaloLeitura*60).toString());
73      });
74  });
```

Listagem 13 – Enviando requisições do usuário para o CloudMQTT.

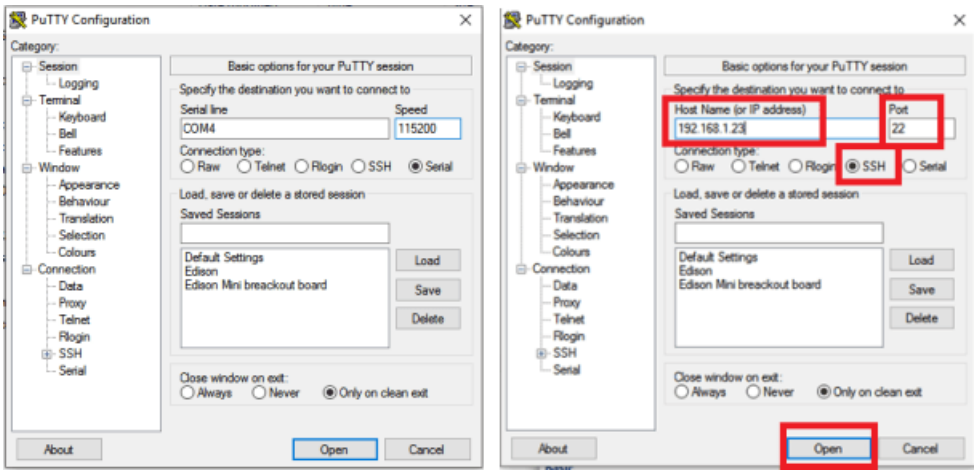
O CloudMQTT permite utilizar a conta cadastrada para acessar a página do *broker* online e visualizar um log de publicações recebidas em cada tópico e baseado no fluxo de informações entre servidor e módulo em campo foi possível atestar que as funcionalidades propostas pro servidor funcionaram como planejado.

Com o servidor finalizado, seguiu-se para o próximo objetivo do projeto: embarcar o sistema para operar diretamente em um dispositivo IoT de baixo custo. Como citado anteriormente, a escolha foi de utilizar o dispositivo Intel Edison e o fabricante disponibiliza para *download* uma ferramenta que realiza toda a configuração inicial do mesmo, deixando o dispositivo pronto para uso (INTEL, 2018a). Essa ferramenta instala todos os drivers necessários para reconhecer o dispositivo no seu computador; realiza a instalação (*Flash Firmware*) do sistema operacional Yocto Linux; atribui uma senha pessoal para a placa para que se habilite o SSH (protocolo *Secure Shell*), permitindo que o dispositivo seja acessado de forma remota e segura pela internet; e por fim estabelece a conexão com a internet nas redes WiFi detectadas. Caso a rede não esteja disponível no momento do setup ou precise ser alterada posteriormente, também é possível realizar uma nova conexão com a internet dentro do terminal do Edison, seguindo as instruções do comando "configure_edison --wifi".

Uma vez finalizado o setup, considerando que o Intel Edison já vem com o Node.js nativamente instalado, foi necessária apenas a instalação do MongoDB que, assim como na máquina virtual, pode ser obtido em uma versão compatível no site oficial da organização.

Para isso é preciso acessar o sistema do dispositivo, e a solução mais comum é utilizar o *software* PuTTY que permite uma conexão serial através da porta COM referente ao dispositivo (se este estiver conectado no computador através de cabo) ou SSH através do endereço IP e porta do dispositivo na rede. A Figura 17 ilustra como ambas as conexões podem ser feitas através da ferramenta PuTTY. Logo, com o Edison ligado e a conexão via PuTTY estabelecida tem-se acesso ao terminal do Linux dentro da placa, e por ele é possível instalar e iniciar o serviço do MongoDB.

Figura 17 – Conexão PuTTY por Serial e por SSH, respectivamente.



Por fim, só foi preciso trazer o projeto do servidor desenvolvido na máquina virtual para dentro do dispositivo, operação feita com o *upload* do diretório do projeto para dentro do Edison. Para isso pode-se utilizar o *software* WinSCP, uma ferramenta bem versátil que permite acesso a memória interna de dispositivos ao se fornecer o endereço IP, porta e autenticação do sistema operacional, e com ela é possível copiar os arquivos diretamente do computador para um diretório do sistema operacional da placa.

Com isto servidor está pronto para execução dentro do dispositivo, mas antes do primeiro uso é necessário fazer o download das demais dependências do sistema. A vantagem nesta etapa é que no diretório do projeto já existem todas as bibliotecas usadas listadas nas dependências do package.json, assim o Gerenciador de Pacotes do Node é capaz de ler o arquivo e instalar todo o necessário para executar a aplicação, como visto a seguir.

```
1 // NOTA: Ambos os comandos devem ser executados dentro do diretorio do
      projeto no Edison
2 //Comando que le o arquivo package.json e instala as dependecias
      listadas
3 $ npm install
4
5 //Comando que inicia o servico do servidor da aplicacao
6 $ npm start
```

Para novos testes foi necessário uma janela do terminal executando o serviço do MongoDB e outra executando o servidor Node.js, permitindo acessar a aplicação web através do endereço "http://[ip_edison]:[porta]/". Com o servidor executando dentro do Intel Edison como proposto, foi feita uma nova checagem das funcionalidades do sistema para verificar se a comunicação entre o servidor e os módulos permanecia em funcionamento. Assim, utilizando um módulo sensor e atuador em ambiente controlado, foi iniciada a coleta de dados para conferir a persistência das leituras no banco de dados do servidor e executadas as funções da aplicação web para interagir com os módulos para testar se o Edison conseguiria atender as requisições, obtendo uma reposta positiva. Mesmo em seu perfeito funcionamento, porém, não foi possível realizar testes unitários de desempenho do servidor (como latência de resposta, gargalo de processamento, entre outros) em tempo viável para o desenvolvimento dessa monografia, mantendo o interesse de uma avaliação mais aprofundada em trabalhos futuros.

4 RESULTADOS E DISCUSSÕES

Para os testes do servidor como solução para um sistema de irrigação automático, foi simulado o funcionamento do sistema como um todo utilizando o servidor de aplicação no Intel Edison e um módulo ESP8266 sensor/atuador (desenvolvido fora do escopo deste trabalho) com sensores inseridos em um vaso com uma porção de terra preta, ambos conectados na internet em ambiente controlado. O módulo sensorial inclui sensores de temperatura e umidade do solo; temperatura, umidade e luminosidade do ambiente e umidade da folha.

Inicialmente foram testadas as interações entre as requisições do servidor e o módulo em campo. Nesse quesito, foi realizado o acesso a aplicação do Edison via navegador e utilizadas as funções do usuário que interagem com o módulo (solicitar leitura, solicitar ativação/desativação e reconfigurar), passando requisições (e dados pertinentes) em tempo real para o servidor através do *socket*. De acordo com a requisição, o servidor é capaz de publicar os dados em um tópico específico do CloudMQTT e a partir dele o módulo ESP consegue responder de acordo com o tópico assinado. Foi possível conferir se o fluxo de mensagens entre servidor e módulo estava funcionando corretamente pois ao acessar o CloudMQTT no navegador temos acesso ao histórico de mensagens obtidas pelo *broker* em cada tópico.

Outro aspecto importante do sistema de irrigação automática é a persistência das leituras. Para isso permitiu-se ao módulo ESP executarsua rotina de coleta de dados e envio de leitura para o servidor ao longo do dia 01 de novembro de 2017 (iniciado às 5h00 e finalizado às 23h30). Neste processo ocorre o inverso das requisições do usuário, agora é o módulo ESP que publica mensagens no tópico de leituras do CloudMQTT que repassa esses dados ao nosso servidor assinante. Assim, os dados dos sensores são inseridos em seu atributo correspondente de um novo objeto do tipo Leitura até que todos os valores sejam preenchidos e então a leitura é salva no banco de dados.

Nesse quesito emerge uma importante discussão referente a confiabilidade da transmissão dos dados. Visto que os testes sempre ocorreram em ambiente controlado não houve nenhum caso de queda da conexão e perda de informações, mas é sabido que se tratando de um sistema de agricultura de precisão pode acontecer casos de uso em áreas rurais cujo sinal seja precário e ocorra interferências no sinal WiFi, o que levanta a oportunidade de aprofundamento no assunto em futuros estudos quanto aos protocolos de rede e soluções em casos de queda de conexão mais frequentes.

É importante destacar positivamente a função do servidor quanto ao monitoramento e gerenciamento dos módulos sensores. Em um caso de uso real em que os módulos sensores podem estar inseridos e espalhados em distantes campos de agricultura, é de grande relevância a capacidade do servidor de manipular e até mesmo configurar o perfil

de funcionamento do módulo remotamente, pelo navegador. Além disso, a visualização dos dados das leituras através de gráficos permite uma melhor compreensão sobre o comportamento dos módulos em uma determinada região. Na Figuras 18 e 19 temos, respectivamente, um panorama geral da variação de temperatura e umidade na região do módulo sensor e na Figura 20 a variação de luminosidade, todas obtidas em ambiente controlado quando foram realizados os testes de coleta de dados.

Figura 18 – Gráfico de leituras de Temperatura do servidor.

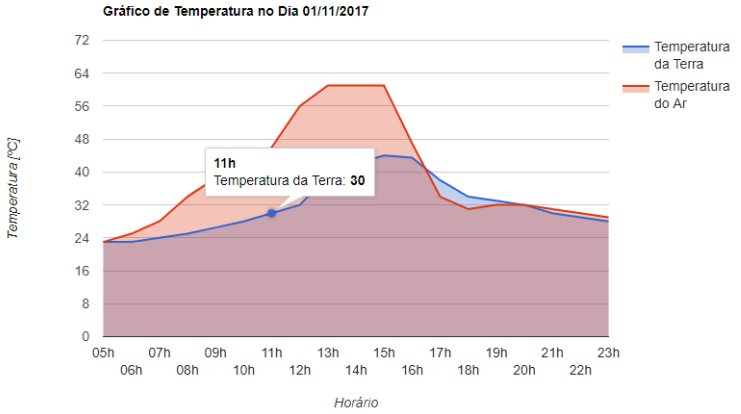
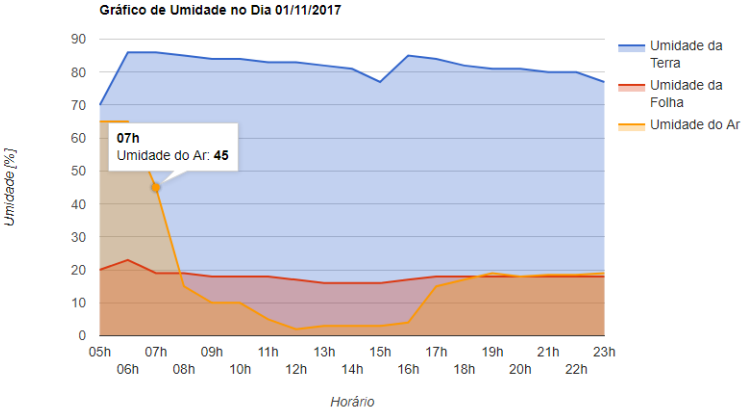
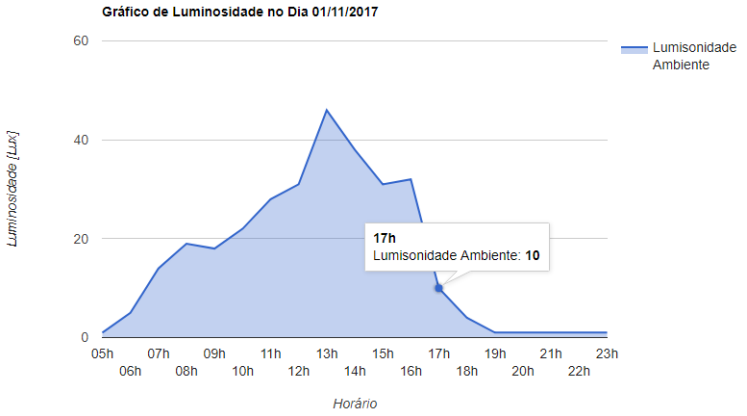


Figura 19 – Gráfico de leituras de Umidade do servidor.



Quanto ao funcionamento da solução embarcada em um dispositivo IoT, foi observado que o Intel Edison respondeu bem aos testes realizados no servidor. Em aplicações que demandam milhares de conexões e requisições concorrentes é possível que se atinja o gargalo de processamento da placa rapidamente, porém, como esses tipos de sistemas de agricultura normalmente são de natureza privada (acessados por somente uma pessoa ou equipe que monitora a plantação) acreditamos que o hardware do dispositivo é suficiente para atender as necessidades, dispensando o investimento em equipamentos robustos de alta despesa financeira e evidenciando o custo x benefício de embarcar a aplicação em um dispostivo SoC, como feito com o Intel Edison. Além disso, a alta flexibilidade do servidor

Figura 20 – Gráfico de leituras de Luminosidade do servidor.



desenvolvido com a pilha MEAN permite a fácil inserção de novos módulos na rede sem fio de sensores, garantindo que o sistema seja escalável em casos de expansão da área de agricultura.

5 CONCLUSÕES

Com o crescimento na área de Internet das Coisas impulsionando o desenvolvimento de tecnologias e serviços online mais eficientes, surge na agricultura uma das principais áreas de aplicação da IoT: a agricultura de precisão. Nesse tipo de aplicação é almejado desenvolver equipamentos e ferramentas que apoiem a utilização eficiente dos recursos disponíveis, sejam eles naturais ou não. E é nesse quesito que o presente trabalho se concentra.

Portanto, foi desenvolvido uma aplicação servidor/cliente como solução para um sistema de irrigação automática cuja principal preocupação foi manter o baixo custo de produção. Para isso, tanto o servidor quanto a aplicação web foram inteiramente implementados com *frameworks* de código aberto e o sistema embarcado em um dispositivo IoT de baixo custo financeiro e consumo de energia em relação a equipamentos mais robustos.

A solução apresentada fornece um servidor que interage com módulos em uma rede de sensores de uma plantação através de um *broker* da internet. Através deste servidor podemos persistir os dados de leituras, criar perfis de plantações e interagir remotamente com os módulos no campo de produção e, devido sua escalabilidade, é possível utilizar o modelo de sistema proposto em diferentes escalas e culturas.

Foram realizados testes das funcionalidades do servidor em ambiente controlado onde foi constatado o correto funcionamento quanto ao fluxo de informações e serviços oferecidos ao usuário. O servidor foi capaz de armazenar os dados enviados pelos sensores e exibi-los ao usuário de forma a permitir a visualização do estado do ambiente e do solo em que o módulo sensor se encontrava. É preciso considerar as limitações do *hardware* de dispositivos SoC, mas devido a natureza privada desse tipo de aplicação concluímos que embarcar o sistema em uma placa como Intel Edison representa um custo x benefício satisfatório.

REFERÊNCIAS

ABOUZAR, P.; MICHELSON, D. G.; HAMDI, M. Rssi-based distributed self-localization for wireless sensor networks used in precision agriculture. **IEEE Transactions on Wireless Communications**, v. 15, n. 10, p. 6638–6650, Oct 2016. ISSN 1536-1276.

ARDUINO AG. Getting started with arduino and genuino products. 2018. Disponível em: <https://www.arduino.cc/en/Guide/HomePage>. Acesso em: 15 nov. 2017.

BOOTSRAP. Getting started. 2018. Disponível em: <http://getbootstrap.com/getting-started/>. Acesso em: 15 jan. 2018.

CHANIOTIS, I. K.; KYRIAKOU, K. D.; TSELIKAS, N. D. Is node.js a viable option for building modern web applications? a performance evaluation study. **Computing**, v. 97, n. 10, p. 1023–1044, Oct 2015. ISSN 1673-5447.

FLORES, K. O. et al. Precision agriculture monitoring system using wireless sensor network and raspberry pi local server. **IEEE Region 10 Conference (TENCON)**, p. 3018–3021, Nov 2016. ISSN 2159-3450.

GOOGLE, INC. Angularjs - superheroic javascript mvw framework. 2018. Disponível em: <https://angularjs.org/>. Acesso em: 29 mar. 2018.

GOOGLE, INC. Charts | google developers. 2018. Disponível em: <https://developers.google.com/chart/>. Acesso em: 12 mai. 2018.

HEO, Y. J. et al. A lightweight platform implementation for internet of things. **Future Internet of Things and Cloud (FiCloud)**, p. 526–531, Aug 2015. ISSN 1555-6719.

IBM. Conhecendo o mqtt. 2018. Disponível em: <https://www.ibm.com/developerworks/br/library/iot-mqtt-why-good-for-iot/index.html>. Acesso em: 19 mai. 2018.

INTEL. DocumentaÇÃo do mÓdulo intel® edison. 2018. Disponível em: <https://software.intel.com/pt-br/iot/hardware/edison/documentation>. Acesso em: 12 mai. 2018.

INTEL. Hardware guide - intel® edison compute module. 2018. Disponível em: <http://download.intel.com/support/edison/sb/edisonmodule_hg_331189004.pdf>. Acesso em: 8 mar. 2018.

JSON.ORG. Introducing json. 2018. Disponível em: <http://json.org>. Acesso em: 23 nov. 2017.

LIBUV CONTRIBUTORS. libuv documentation. 2018. Disponível em: <http://docs.libuv.org/en/v1.x/>. Acesso em: 21 mar. 2018.

LOZOYA, C.; AGUILAR, A.; MENDOZA, C. Service oriented design approach for a precision agriculture datalogger. **IEEE Latin America Transactions**, v. 14, n. 4, p. 1683–1688, April 2016. ISSN 1548-0992.

MEAN.IO. Home. 2018. Disponível em: <http://mean.io/>. Acesso em: 21 mar. 2018.

MICROSOFT. Model-view-controller. 2018. Acesso em: 21 jan. 2018.

MONGODB, INC. What is mongodb? 2018. Disponível em: <https://www.mongodb.com/what-is-mongodb>. Acesso em: 18 mai. 2018.

NODE.JS FOUNDATION. About node.js. 2018. Disponível em: <https://nodejs.org/en/about/>. Acesso em: 15 jan. 2018.

NODE.JS FOUNDATION. Home. 2018. Disponível em: <http://expressjs.com>. Acesso em: 07 mai. 2018.

PEREIRA, C. R. **Node. js: Aplicações Web Real-Time com Node. js**. Primeira. [S.l.]: Casa do Código, 2013.

POULTER, A. J.; JOHNSTON, S. J.; COX, S. J. Using the mean stack to implement a restful service for an internet of things application. **IEEE World Forum on Internet of Things**, Dec 2015. ISSN 1572-9204.

RASPBERRY PI FOUNDATION. About us. 2018. Disponível em: <https://www.raspberrypi.org/about/>. Acesso em: 15 nov. 2017.

SOCKET.IO. Socket.io documentation. 2018. Disponível em: <https://socket.io/docs/>. Acesso em: 19 mai. 2018.

SUPALLA, Z. Introduction to the guide. 2018. Disponível em: <https://docs.particle.io/guide/getting-started/intro/photon/>. Acesso em: 15 nov. 2017.

YIN, S. et al. Design of wireless multi-media sensor network for precision agriculture. **China Communications**, v. 10, n. 2, p. 71–88, March 2013. ISSN 1673-5447.

ZHOU, L. et al. Roscc: An efficient remote sensing observation-sharing method based on cloud computing for soil moisture mapping in precision agriculture. **IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing**, v. 9, n. 12, p. 5588–5598, Dec 2016. ISSN 1939-1404.