UNIVERSIDADE FEDERAL DE VIÇOSA
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ANDRÉ LOBO TEIXEIRA

DESENVOLVIMENTO DE UM CLIENTE OPC E UMA WEB APPLICATION PARA DISPONIBILIZAR INFORMAÇÕES DE PLANTAS INDUSTRIAIS NA INTERNET

VIÇOSA
2017

ANDRÉ LOBO TEIXEIRA

DESENVOLVIMENTO DE UM CLIENTE OPC E UMA WEB APPLICATION PARA DISPONIBILIZAR INFORMAÇÕES DE PLANTAS INDUSTRIAIS NA INTERNET

Monografia apresentada ao Departamento de Engenharia Elétrica do Centro de Ciências Exatas e Tecnológicas da Universidade Federal de Viçosa, para a obtenção dos créditos da disciplina ELT 490 – Monografia e Seminário e cumprimento do requisito parcial para obtenção do grau de Bacharel em Engenharia Elétrica.
Orientador: Prof. Dr. André Gomes Torres.
Co-orientadora: Profa. Dra. Kétia Soares Moreira.

VIÇOSA
2017

ANDRÉ LOBO TEIXEIRA

DESENVOLVIMENTO DE UM CLIENTE OPC E UMA WEB APPLICATION PARA DISPONIBILIZAR INFORMAÇÕES DE PLANTAS INDUSTRIAIS NA INTERNET

Monografia apresentada ao Departamento de Engenharia Elétrica do Centro de Ciências Exatas e Tecnológicas da Universidade Federal de Viçosa, para a obtenção dos créditos da disciplina ELT 490 – Monografia e Seminário e cumprimento do requisito parcial para obtenção do grau de Bacharel em Engenharia Elétrica.

Aprovada em 05 de dezembro de 2017.

COMISSÃO EXAMINADORA

Prof. Dr. André Gomes Torres - Orientador
Universidade Federal de Viçosa - DEL

Profa. Dra. Kétia Soares Moreira – Co-orientadora
Universidade Federal de Viçosa - DEL

Prof. Dr. André Gustavo dos Santos - Membro
Universidade Federal de Viçosa - DPI

“Quem, de três milênios,
Não é capaz de se dar conta.
Vive na ignorância, na sombra,
À mercê dos dias, do tempo.”
(Johann Wolgang von Goethe)

Aos meus pais Ary e Ivanete.

Agradecimentos

Agradeço a todos que estiveram comigo, perto ou longe, e me ajudaram em mais uma vitória.

Agradeço aos meus pais, Ary e Ivanete, pelo amor e dedicação à minha vida, vocês são responsáveis por tudo o que já conquistei. Obrigado pelo apoio nas horas difíceis e por sempre acreditarem em mim.

Agradeço aos amigos da república e agregados, parte vital para a motivação durante a execução deste trabalho. Agradecimento especial ao Jhonatan, por sempre ajudar nas soluções para os problemas encontrados neste projeto.

Agradeço ao meu orientador André Gomes Torres, pela oportunidade de realizar um projeto em uma área que tenho no coração.

Gostaria de agradecer ao Nicolae Brinzei, pela excelente oportunidade de ir além neste trabalho.

Gostaria de agradecer à ENSEM e a UFV, devido ao acordo entre as instituições sem o qual não seria possível aprimorar este trabalho.

Resumo

Uma ferramenta central no processo de tomada de decisões nas indústrias é a informação, portanto facilitar o seu acesso pode trazer benefícios para a gestão e controle dos processos envolvidos. O trabalho em questão se encontra no contexto de supervisão industrial, objetivando o desenvolvimento de um sistema que disponibilize informações de dispositivos de plantas industriais, para que estas sejam acessadas de qualquer lugar por meio da internet. Visando aumentar a versatilidade de aplicação e utilização do sitema, foi utilizado um padrão largamente difundido na indústria para acesso às informações das plantas: o padrão OPC. Além disso, a aplicação que disponibiliza os dados foi desenvolvida para ser acessível por qualquer dispositivo que tenha acesso à Internet e possa utilizar um navegador *web*. Pela análise dos resultados, o sistema mostrou-se capaz de realizar seu objetivo em cenários variados, podendo fornecer a troca de informações a grandes distâncias e em um tempo consideravelmente pequeno, além de poder ser aplicado em uma planta dinâmica, realizando um controle PID em um motor de corrente contínua.

Abstract

A central tool in the decision-making process in industries is information. Therefore, facilitating their access can bring benefits to the management and control of the processes involved. This work is in the context of industrial supervision, aiming the development of a system that provides information of industrial plant devices, so that they can be accessed from anywhere through the internet. In order to increase the versatility of the application and use of the system, a widespread industry standard has been used to access plant information: the OPC standard. In addition, the application that makes the data available is designed to be accessible by any device that has Internet access and can use a web browser. By the analysis of the results, the system was able to accomplish its objective in varied scenarios, being able to provide the information exchange at great distances and in a considerably small time, besides being able to be applied in a dynamic plant, performing a PID control in a DC motor.

Sumário

Lista de Figuras

1 Introdução

Uma característica marcante do início do presente século é o grande avanço tecnológico nas mais diferentes áreas. Tecnologias que antes eram caras, inacessíveis ou até inexistentes estão nas mãos de pessoas comuns, desde crianças a adultos. Além disso, as pessoas estão o tempo inteiro conectadas umas às outras por meio da internet, podendo se comunicar em tempo real com alguém a centenas de quilômetros. A informação encontra-se à distância de um clique e, em muitos casos, de forma gratuita.

Trazer para a indústria esta facilidade do acesso à informação provida, em grande parte, pela internet é um grande desafio. Porém, dado que a informação é ferramenta central no processo de tomada de decisões em indústrias, facilitar o seu acesso pode trazer diversos benefícios para a gestão das organizações, sendo, portanto, uma boa motivação para os desafios existentes **[1].**

Dispositivos, aplicativos ou *softwares* capazes de monitorar, atuar e gerar relatórios de forma instantânea podem otimizar processos industriais, aumentar a produtividade, reduzir custos e desperdício, proporcionando maior segurança e uma melhor gestão. Um sistema que disponibilize informações de dispositivos de plantas industriais, para que estas sejam acessadas de qualquer lugar por meio da internet, permite identificar e solucionar problemas de forma rápida.

Nas seções seguintes serão descritos alguns conceitos teóricos importantes para o desenvolvimento deste trabalho, bem como o objetivo do mesmo. No capítulo 2 são detalhados os materiais utilizados para a contrução do sistema proposto, bem como a metodologia de desenvolvimento dos *softwares* que integram o sistema. No capítulo 3 são apresentados os resultados e discussões obtidos ao se aplicar o sistema. No capítulo 4 é apresentada a conclusão do trabalho.

1.1 Programação orientada a objetos

Um importante paradigma de programação é a Programação Orientada a Objetos (POO). Por paradigma denomina-se a forma em que se pensa para se estruturar um programa de computador, além da sua natureza de execução. O grande poder fornecido pela POO é o desenvolvimento de programas vendo-os como a junção de vários objetos. Estes objetos possuem propriedades e podem realizar certas ações. Desta forma, o raciocínio que se emprega é mais próximo do mundo real em que os seres humanos experimentam, facilitando o desenvolvimento do programa **[2].**

Por propriedades e ações de um objeto pode-se exemplificar, respectivamente, o nome e o ato de falar de uma pessoa. A pessoa, objeto no caso, possui um nome que é uma de suas propriedades, além de ter a capacidade de executar a ação de falar. A propriedade "nome" pode ser representada por uma *string*, enquanto a ação "falar" poderia, por exemplo, ser representada por uma função que retorna uma *string*.

Por meio deste simples raciocínio pode-se introduzir um dos conceitos fundamentais da POO: as classes. As classes são os modelos para criação dos objetos. Nelas encontram-se definidas quais as propriedades, também chamadas de atributos, o objeto possui, bem como o tipo destes atributos. Nas classes encontram-se, também, ações que podem ser executadas por este objeto, referidas como métodos. O principal método de uma classe é chamado de "construtor", por meio do qual pode-se efetivamente criar os objetos, os quais serão referidos como instâncias das classes **[3].**

A POO foi constantemente empregada na execução deste trabalho, principalmente auxiliando no desenvolvimento do *software* descrito na subseção 2.2.1.

1.2 Threads

Threads são unidades independentes de um processo, sendo este último também conhecido como programa. As *threads* podem ser vistas como subprocessos leves, geralmente criadas para executar tarefas específicas **[4].**

Existem programas que possuem diversas funções, como mostrar interfaces gráficas na tela, processar cálculos e aguardar entradas do usuário. A utilização de *threads* para

programas com estas características não se trata somente da organização das funções, mas também de um melhor aproveitamento do uso da CPU. Por exemplo, uma *thread* específica para esperar entradas do usuário, isolando esta tarefa, evita que a execução de outras funções seja prejudicada **[5].**

1.3 Padrão OPC

A interoperabilidade de um sistema é a sua capacidade de se comunicar sem restrições com outros sistemas, sejam estes semelhantes ou não. Dado este conceito, o *Object Linking and Embedding for Process Control* (OPC) é um padrão criado pela OPC *Foundation* com objetivo de promover a comunicação de dispositivos de campo, PLC's, interfaces homem-máquina e outros sistemas, os quais são comumente encontrados em uma rede industrial **[6]**.

As primeiras especificações desenvolvidas, entre os anos de 1996 e 2001, foram baseadas nas tecnologias proprietárias da Microsoft® conhecidas como *Object Linking and Embedding* (OLE) e *Distributed Component Object Model* (DCOM). Por este motivo, as aplicações que poderiam ser desenvolvidas restringiam-se ao sistema operacional Windows. Contudo, em 2003, a OPC Foundation lançou as especificações para o OPC *Unified Architecture* (OPC UA), o que fez o padrão se extender a outros sistemas operacionais **[7].**

1.4 Web Application

Web applications são aplicações que podem ser acessadas através da Internet por meio de um *web browser*, como os conhecidos *Google Chrome* e *Mozilla Firefox*. O desenvolvimento de *web applications* envolve a combinação de diversas tecnologias, geralmente referenciadas como uma pilha de tecnologias de *software* em que cada uma oferece uma funcionalidade.

As funcionalidades que podem ser citadas são servidores que possuem a capacidade de resolver requisições em HTTP, um tipo de banco de dados para armazenar informações e uma linguagem em que se possa escrever código para gerenciar as interações do usuário com o *web browser*. Um exemplo de tecnologias difundidas para compor os elementos de uma *web application* é o servidor *web Apache*, o banco de dados relacional *MySQL* e a linguagem PHP **[8].**

É necessário que o programador escolha e domine todas as tecnologias da pilha, além de saber interconectá-las de maneira satisfatória, manejando bem os princípios e restrições existentes. Este processo pode ser, por diversas vezes, custoso. Pode tornar-se viável, portanto, a utilização de ferramentas que disponibilizem a pilha de tecnologias e forneçam métodos para o desenvolvimento da *web* application.

Neste contexto, pode-se citar o *Meteor*, ferramenta por meio da qual será desenvolvida a *web application* existente neste trabalho, o qual será descrito na subseção 2.1.5 **[9].**

1.5 Objetivo Geral

O Objetivo geral deste trabalho é o desenvolvimento de um sistema que disponibilize informações de dispositivos de plantas industriais, para que estas sejam acessadas de qualquer lugar por meio da internet. O objetivo geral pode ser dividido nos seguintes objetivos:

- Desenvolver uma aplicação que seja capaz de ser um cliente OPC, podendo, assim, requisitar dados dos dispositivos das plantas industriais aos servidores OPC;
- Incluir uma interface gráfica na aplicação para facilitar a seleção dos dispositivos que se deseja monitorar, além de providenciar informações em tempo real ao usuário;
- Criar na aplicação um mecanismo de comunicação com um banco de dados na nuvem, o qual será o intermediário para a disponibilização das informações na internet;
- Desenvolver uma *Web application* que possa se conectar ao banco de dados e monitorar as informações disponibilizadas pelo cliente OPC;

2 Materiais e Métodos

Neste capítulo, serão descritos, respectivamente, nas seções 2.1 e 2.2 os materiais e métodos que compõem o desevolvimento deste trabalho.

2.1 Materiais

Nesta seção, são listadas e brevemente descritas as ferramentas utilizadas para desenvolver as aplicações existentes no sitema, para providenciar o banco de dados e importantes bibliotecas que permitiram a integração entre os vários elementos.

2.1.1 Visual Studio 2012

O *Visual Studio 2012* é um *Integrated Development Environment* (IDE), ou seja, um *software* que integra diversas ferramentas úteis para desenvolvimento, como por exemplo editores de código, compiladores, interpretadores e depuradores. Na maioria dos casos, o ambiente possui uma *Graphical User Interface* (GUI), a qual auxilia o programador a utilizar as ferramentas e, inclusive, gerenciar e produzir seu código de maneira mais eficiente **[10].**

Na IDE utilizada, pode-se, através de sua GUI, gerenciar as propriedades iniciais dos objetos e classes criadas, assim como definir localizações dos componentes de uma GUI que se queira criar para a aplicação. Desta forma, agiliza-se o processo de desenvolvimento do *software*, focando-se mais em suas funcionalidades e reduzindo a quantidade de código que o programador deve escrever **[11].**

Com o *Visual Studio 2012* pode-se criar aplicações em linguagens como C++ e C#, além de se criar projetos em que a IDE orienta a casos mais específicos como *Console Application* e *Windows Forms Application*. Nesta última opção, pode-se criar uma aplicação que possui interface gráfica, utilizando a biblioteca gráfica da *.NET*, a qual pode ser acessada pelo *Namespace "System.Windows.Forms"*.

Um outro aspecto importante que a IDE auxilia e será utilizado neste trabalho é a inserção de referências a bibliotecas dinâmicas, como por exemplo aquelas com extensão *.dll*.

Desta forma, pode-se usar as classes e métodos definidos nestes arquivos, como foi feito para se usar as ferramentas que serão descritas nas duas subseções seguintes: a biblioteca para se comunicar com o banco de dados *MongoDB* e a biblioteca para se utilizar o padrão OPC.

2.1.2 *Advosol* OPC DA .NET Client Component

A *Advosol Inc.* é uma empresa que desenvolve *softwares* e ferramentas para uso do padrão OPC, tendo começado a oferecer este tipo de serviço no ano de 1996, mesmo ano de lançamento do padrão OPC DA. Um de seus produtos é o pacote de desenvolvimento de clientes OPC para plataforma .NET, o qual é indicado no site da OPC *Foundation* como referência para desenvolvimentos de clientes OPC DA em linguagem *C#* ou *VB.NET* **[12, 7].**

Este pacote disponibiliza vários utensílios que podem ajudar no desenvolvimento de aplicações que envolvem o padrão OPC. Dentre estes tem-se clientes e servidores OPC para testes, *softwares* para auxiliar na descoberta de problemas ao se conectar a servidores e a biblioteca dinâmica chamada *OpcDaNet.Net4.dll*. Por meio desta última pode-se, por exemplo, utilizar classes implementadas em *C#* para realizar operações de leitura e escrita em dispositivos configurados em um servidor OPC **[13].**

Foi utilizada, neste trabalho, uma licença de teste que é gratuita, tendo como limitação o tempo de trinta minutos após o uso da classe que faz a conexão com os servidores OPC.

2.1.3 *MongoDB .NET driver*

Uma outra ferramenta usada como biblioteca dinâmica foi o *MongoDB .NET driver*, a qual é indicada no site oficial do *MongoDB* **[14]**. Nesta biblioteca, obtida em **[15]**, são encontradas classes implementadas em *C#* para se fazer as manipulações em um banco de dados do tipo *MongoDB*.

MongoDB é um banco de dados não relacional onde se criam coleções que armazenam documentos no formato *BSON*, o formato binário do *JavasScript Object Notation* (JSON). Os documentos possuem, portanto, elementos do tipo chave-valor. O *MongoDB* é um banco de dados orientado a documentos e projetado para ser flexível, rápido e suportar esquemas ricos e dinâmicos **[16][17]**.

Podem ser encontrados bancos de dados *MongoDB* na internet sendo oferecidos como serviços, como foi utilizado neste trabalho e será descrito na subseção seguinte.

2.1.4 *mLab*

Atualmente, o *mLab* gerencia mais de meio milhão de bancos de dados em todo o mundo, com ajuda de grandes conhecidos fornecedores de nuvem como a *Amazon Web Services*, *Microsoft Azure* e *Google*. O *mLab* é, portanto, um serviço de banco de dados de nuvem, especificamente para bancos não relacionais do tipo *MongoDB*. Disponibiliza ferramentas de gerenciamento, monitoramento e backup, além de suporte de especialistas **[18].**

Por meio do site é possível gerenciar o banco de dados, tendo-se acesso aos documentos e podendo-se, inclusive, editá-los conforme desejado. Uma outra possibilidade que pode se mostrar útil é a exportação e importação das coleções, podendo esta ser feita usando formatos *.csv*. Isto abre a possibilidade de se explorar os dados registrados de maneira simples, usando *softwares* como por exemplo o *MATLAB*® e o *Excel*®.

Ao se criar uma conta no *mLab*, pode-se obter, gratuitamente, um banco de dados para testes que disponibiliza um espaço de 500Mb. É providenciada, então, uma *Uniform Resource Identifier* (URI), uma *string* que possui os identificadores necessários para realizar a conexão com o banco de dados pela aplicação que se desejar. Desta maneira, será possível usar o banco de dados em questão na aplicação em *C#* e na *Web Application*, sendo esta última criada com a ferramenta que será descrita na subseção seguinte.

2.1.5 Meteor

O *Meteor* é um conjunto de tecnologias e métodos que possibilitam o desenvolvimento de *web applications*. É comumente referenciado como uma ferramenta de desenvolvimento *full-stack*, porque a mesma dá suporte e define tecnologias para desenvolvimento do *front-end*, *back-end* e banco de dados.

Portanto, com esta ferramenta será possível criar uma *web application* completa, desde a parte do cliente até o servidor, além de realizar a sua integração com o banco de dados *MongoDB* criado com o *mLab*. Uma outra vantagem é a sua compatibilidade com difundidas

tecnologias, como a biblioteca *BootStrap*. Fazendo simples alterações no código HTML, é possível definir o *design* dos compontes que serão dispostos ao usuário da aplicação **[19][20].**

Uma vantagem que pode ser citada desta ferramenta é a possibilidade de se programar tanto a parte do cliente quanto a parte do servidor em somente uma linguagem: *JavaScript*. Desta forma, o desenvolvimento da aplicação é facilitado, podendo-se focar em sua funcionalidade.

2.2 Métodos

Esta seção será dedicada à explicação da arquitetura e do desenvolvimento do sistema, o qual pode ser visto no esquema da **Figura 1**. Neste esquema, são destacadas as aplicações desenvolvidas e as ferramentas, citadas na seção anterior, utilizadas tanto na criação das aplicações, quanto na interconexão dos elementos do sistema.

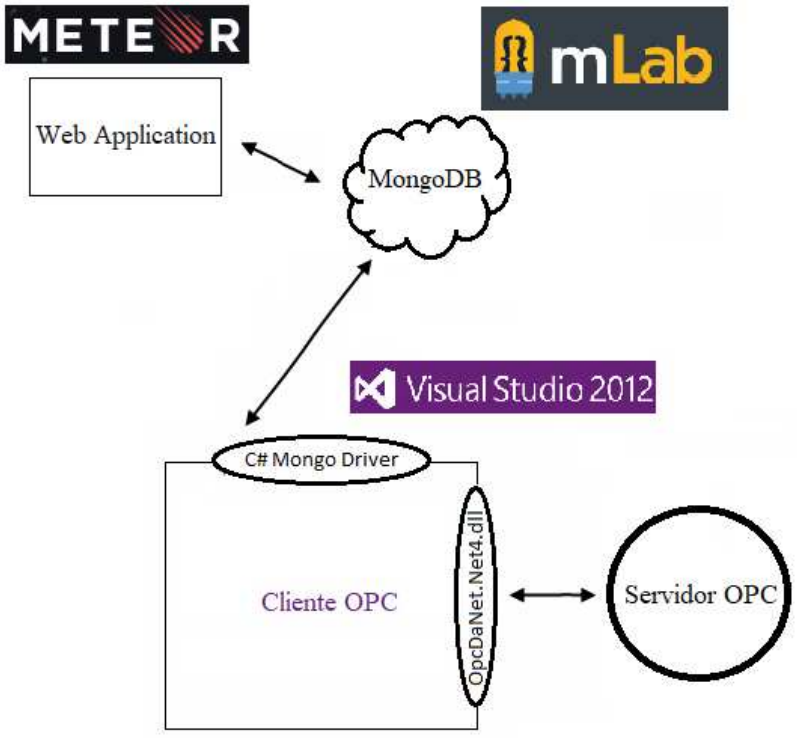


Figura 1. Esquema do sistema e as ferramentas utilizadas.

O sistema pode ser decomposto em três partes: o *Cliente OPC*, o banco de dados *MongoDB* e a *Web Application*. Estas partes são descritas nas subseções seguintes.

2.2.1 Cliente OPC

A aplicação em questão pode ser definida como uma interface entre o banco de dados *MongoDB* e os servidores OPC. Com auxílio de uma interface gráfica do usuário, ela terá o objetivo de dar os seguintes direitos ao controlador:

- Pesquisar por servidores OPC em execução na máquina;
- Conectar-se com algum dos servidores OPC encontrados ou conectar-se com um servidor por meio da inserção manual do seu nome;
- Com auxílio de uma *TreeView*, navegar pelas informações disponibilizadas pelo servidor, como por exemplo os dispositivos definidos no mesmo;
- Definir quais destes dispositivos serão registrados no banco de dados, o que os tornarão visíveis através da *Web Application;*
- Interromper e começar os processos de requisições ao servidor OPC, de verificação da existência de requisições vindas da *Web Application* no banco de dados e de atualização das informações no mesmo.

Os processos enunciados no último item compõem o mecanismo de interface entre o banco de dados e o servidor OPC. Estes foram implementados em uma classe que possui o nome *FormConnection*, a qual será detalhada nas subseções de descrição da interface gráfica do usuário e do mecanismo de interface.

2.2.1.1 Interface gráfica do usuário

Com auxílio do *Visual Basic 2012*, foi criado um projeto do tipo *Windows Forms Application.* A classe *FormConnection* foi definida, sendo que esta herda da classe *Form.* Esta última já existe na plataforma .NET e é utilizada para criar janelas onde se pode incluir diversos componentes como: botões, caixas de textos, *labels* e entre outros.

O *design* da interface foi feito com ajuda do *Visual Basic 2012*, o qual possui ferramentas em que se pode arrastar e posicionar os compontentes, bem como editar as suas propriedades. Automaticamente, são criados os códigos para instanciar os objetos que representam estes componentes, sendo estes definidos como membros da classe

FormConnection. A interface do usuário com todos os componentes gráficos utilizados pode ser vista na Figura 2.

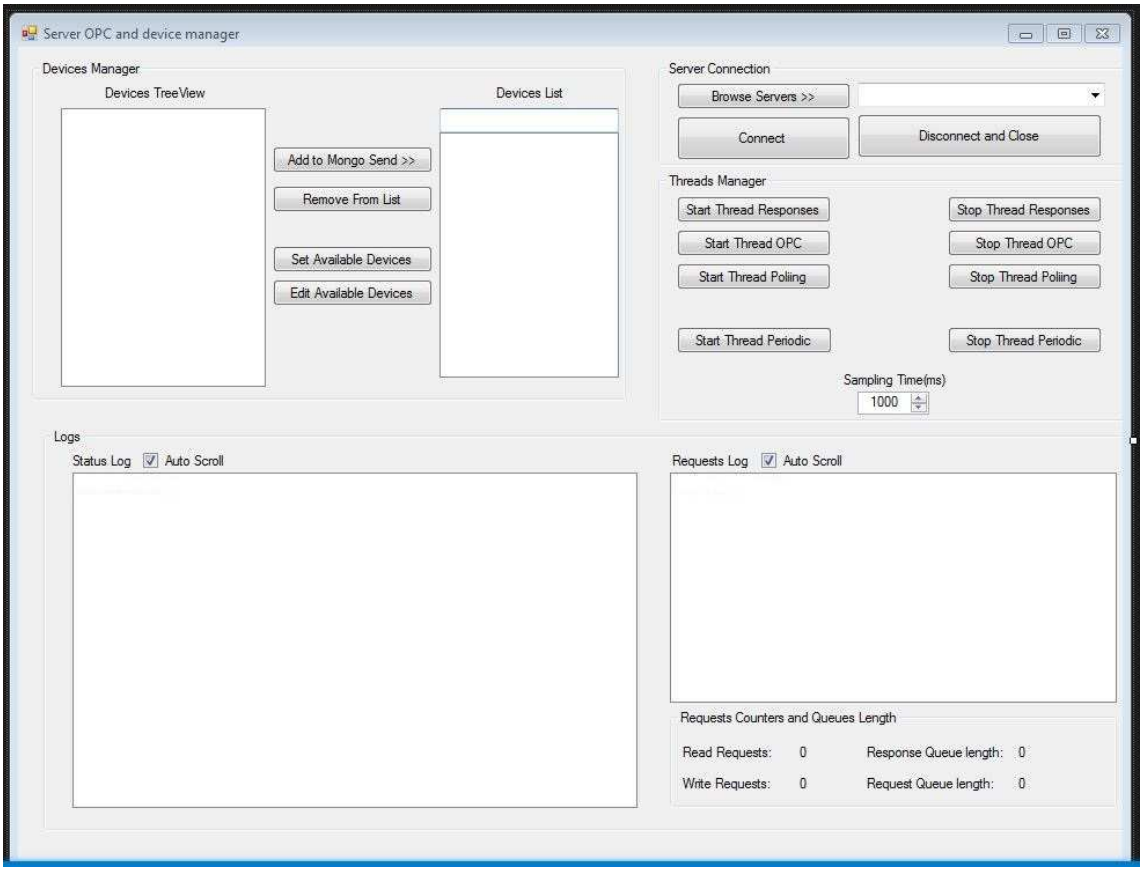


Figura 2. Interface gráfica do Cliente OPC.

Como dito, cada componente do *design* é um membro da classe *FormConnection*, portanto estes podem ser acessados pelos métodos desta classe. Desta forma, alguns dos métodos da classe *FormConnection* são destinados para o tratamento dos eventos que podem ser produzidos pelos componentes da interface, como por exemplo cliques e seleções pelo mouse. Também como membros desta classe, foram definidos os diversos objetos utilizados no mecanismo de interface, como por exemplo: objetos das bibliotecas OPC necessários para realizar a conexão com o servidor, objetos para a conexão com o banco de dados, além de quatro *threads* e duas estruturas de fila, as quais serão detalhados posteriormente.

Portanto, quando a aplicação for executada, ela basicamente criará uma instância de um objeto da classe *FormConnection* e a interface do programa será disponibilizada na tela para o usuário. Nesta interface, o utilizador terá controles a sua disposição, os quais são

agrupados para facilitar a identificação dos seus funcionamentos. Os grupos são denominados: *Server Connection*, *Devices Manager*, *Threads Manager* e *Logs*. Suas funções são descritas a seguir.

Na **Figura 3**, pode ser visto em detalhe o grupo *Server Connection*. Este grupo possui um botão para se fazer a busca dos servidores OPC existentes na máquina e adicionar seus respectivos nomes em um *ComboBox*. Desta forma, é possível escolher o servidor para se conectar e desconectar, com auxílio de outros dois botões. A desconexão também encerra a aplicação, salvando todos os dados relevantes.

Figura 3. Grupo de controles *Server Connection*.

O grupo *Devices Manager*, mostrado na **Figura 4**, contém uma *TreeView*. Por meio deste componente será possível navegar pelos dispositivos existentes no servidor após a conexão com o mesmo. Os dispositivos selecionados nesta *TreeView* poderão ser adicionados ou removidos, com a ajuda dos dois primeiros botões superiores, a uma lista de *strings*, a qual será disposta no *ComboBox* localizado à direita. Outros dois botões serão usados para atualizar estes dispositivos da lista no banco de dados, registrando-os como disponíveis para acesso pela *Web Application*.

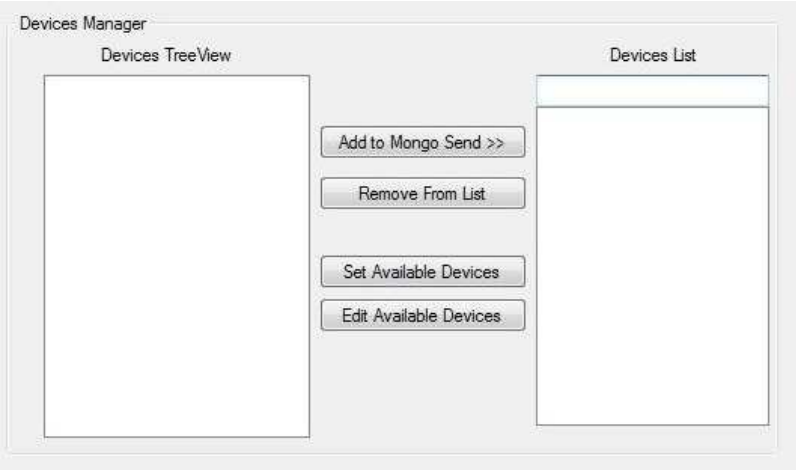


Figura 4. Grupo de controles *Devices Manager*.

Na **Figura 5**, pode-se ver o *Threads Manager*, o qual possui botões que fornecerão ao usuário o controle de começar e parar, individualmente, as quatro principais *threads* do programa. Utilizando um componente do tipo *NumericUpDown*, será possível escolher o tempo de amostragem para o caso de leituras periódicas.

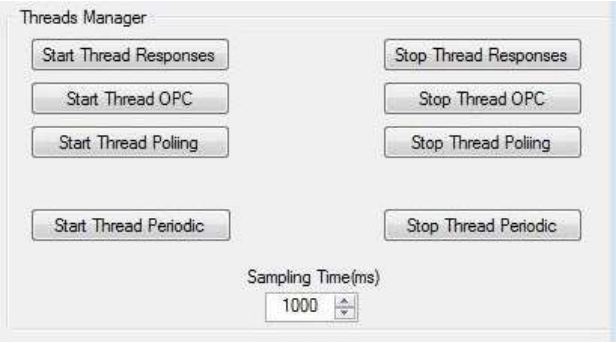


Figura 5. Grupo de controles *Threads Manager*.

Por fim, na **Figura 6**, o grupo *Logs* possui objetos com a finalidade de mostrar informações em tempo real dos eventos ocorridos, especificando a data dos mesmos. Além das diversas situações que poderão ocorrer, como conexão, identificação de pedidos de leitura ou escrita, pedido de desconexão e outros, poder-se-á ver a quantidade de leituras e escritas já executadas e monitorar o tamanho das filas de resposta e de requisições.

Figura 6. Grupo de controles *Logs*.

Ao fim da conexão, estas informações serão gravadas em arquivos estruturados nos formatos *.txt* e *.csv*. Estes arquivos serão importantes tanto para análise do histórico de eventos da conexão, quanto para análise dos dados adquiridos dos dispositivos, como seus valores e suas respectivas datas de aquisição.

A interface gráfica é um elemento importante do *Cliente OPC*. Além de propiciar o controle do mesmo, ela disponibilizará dados ao usuário que possibilitarão o monitoramento dos eventos no tempo e a identificação de eventuais inconsistências.

2.2.1.2 Mecanismo de interface

Como membros da classe *FormConnection*, foram definidas quatro *threads* referenciadas como: *threadOPCAccess*, *thrMongoPolling*, *thrMongoResponse* e *thrPeriodicRead*. Na execução de suas respectivas funções, estas *threads* acessarão duas estruturas de fila que também serão definidas como membros da classe *FormConnection*, referenciadas como *Responses* e *Requests*. Na **Figura 7** são ilustradas as funções de cada *thread*.

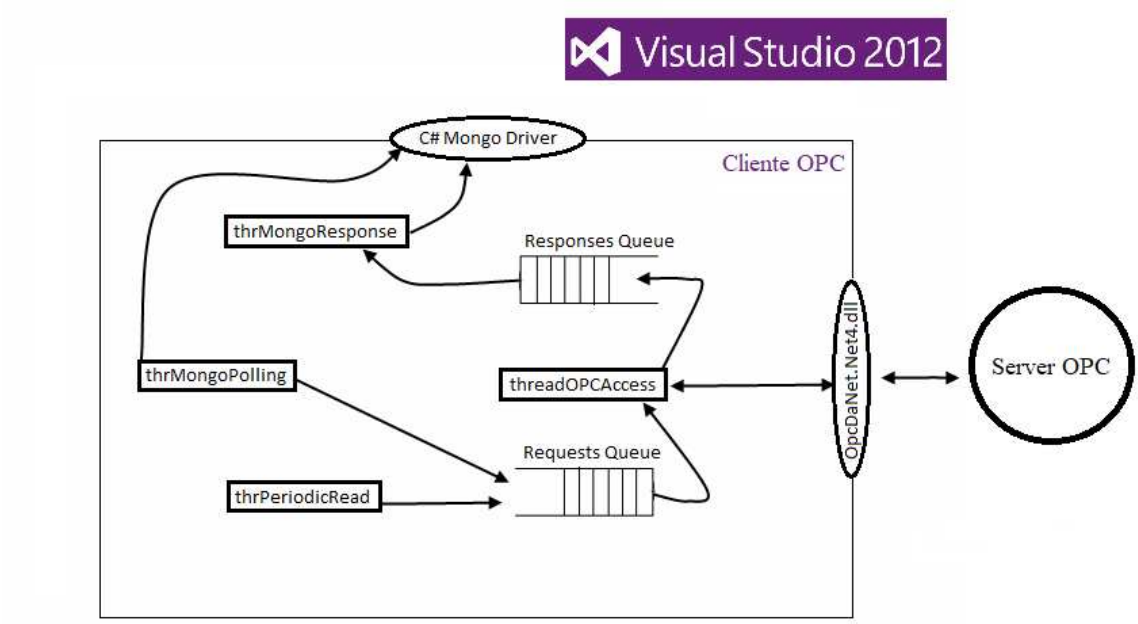


Figura 7. Ilustração das funções das *threads* e das filas.

A *thrMongoPolling* é responsável por verificar continuamente no banco de dados a existência de pedidos de leitura, de escrita ou de início ou parada de leituras periódicas. Ao identificar algum pedido, ela o colocará na fila *Requests*, incluindo um identificador se é pedido de leitura ou escrita. Para os casos de leitura periódica, esta notificará a *thrPeriodicRead*, o que fará com que esta *thread* inicie ou pare sua função. A função desta

última *thread* citada é simplesmente colocar pedidos de leitura na fila *Requests*, sendo isto feito com o período de tempo definido por meio da interface gráfica.

Por sua vez, a *threadOPCAccess* retira os elementos da fila *Requests* e, conforme o seu tipo e utilizando as informações contidas no pedido, fará a requisição ao server OPC. Após obter a resposta do server OPC, o pedido já tratado será colocado na fila *Responses.* O tipo de variável em que os pedidos estão representados é o *BsonDocument*, o qual será detalhado na seção do banco de dados.

Por fim, a *thrMongoResponse* se encarrega de retirar os pedidos da fila *Responses* e fazer a inserção dos mesmos, em seus devidos lugares, no banco de dados. Para evitar o problema de competição entre as *threads* pelas duas estruturas de fila, existirão objetos da classe *Mutex* definidos para cada uma delas. Por meio destes, será assegurado que durante os procedimentos de inserção ou retirada de um elemento das filas, somente um dos processos esteja acessando estes objetos.

2.2.2 Banco de dados

Foram criadas quatro coleções no banco de dados *MongoDB* utilizado, sendo estas: *serversOPC*, *devicesAvailable*, *readRequests* e *writeRequests*. Cada coleção guardará documentos que serão utilizados para representar, respectivamente, informações dos servidores OPC acessados, informações dos dispositivos disponíveis e dos pedidos de leitura e de escrita. Tanto o cliente da *Web Application* quanto o *Cliente OPC* poderão modificar os campos chave-valor destes documentos ao realizarem suas ações.

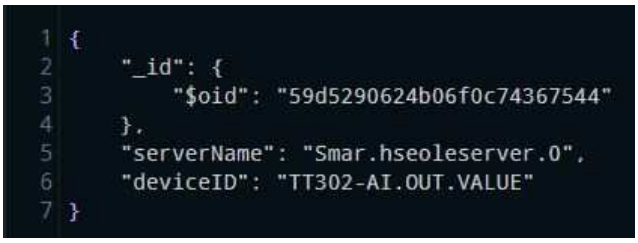
Na **Figura 8**, tem-se o exemplo de um documento da coleção *serversOPC*. Pode-se obter informações do nome do servidor OPC com o campo *serverName*, se está conectado ou não com o campo *connected* e saber qual a data de sua última conexão, registrado no campo *lastTimeOn*. Além disso, existem outros dois campos chamados *checked* e *command*, utilizados para notificar comandos de início ou parada da leitura periódica no servidor. Modificando estes campos, o cliente da *Web Application* poderá acionar ou parar esta leitura, bem como saber se ela está acontecendo ou não.

```
{
    "_id": {
        "$oid": "59d3b665739ae80f3c9a5749"
    },
    "serverName": "Schneider-Aut.OFS.2",
    "connected": false,
    "checked": true,
    "command": "stopped",
    "lastTimeOn": {
        "$date": "2017-10-06T11:19:59.069Z"
    },
    "inMachineNamed": "offline"
}
```


Figura 8. Exemplo de documento da coleção *serversOPC*.

Na coleção *devicesAvailable* serão guardados documentos que representarão os dispositivos definidos como visíveis para o cliente da *Web Application*. Um dos campos destes documentos é o *serverName*, o qual possui como seu valor uma *string* com o nome do servidor a que pertence o dispositivo. Este, por sua vez, é identificado por uma *string* registrada como valor do campo *deviceID*.

Um exemplo de um documento desta coleção é visto na **Figura 9**, onde tem-se um dispositivo identificado por *"TT302-AI.OUT.VALUE"* e pertencente ao servidor OPC *"Smar.hseoleserver.0"*. Estando os dipositivos registrados no banco de dados, o cliente da *Web Application* poderá fazer requições de leitura e escrita aos mesmos. Estas requisições também serão representadas por documentos no banco de dados, conforme a explicação a seguir.



```
{
    "_id": {
        "$oid": "59d5290624b06f0c74367544"
    },
    "serverName": "Smar.hseoleserver.0",
    "deviceID": "TT302-AI.OUT.VALUE"
}
```


Figura 9. Exemplo de documento da coleção *devicesAvailable*.

Na **Figura 10** e na **Figura 11** podem ser vistos, respectivamente, exemplos dos documentos que representam os pedidos de leitura e escrita nos dispositivos. Existem campos que são comuns entre ambos, como o *checked, checkedAt, createdAt, requestT* e *serverName*. Nestes campos são armazenadas informações das *requests* criadas, sendo estas: se ela já foi tratada pelo servidor, a data deste tratamento, a data de sua criação e as *strings* do dispositivo e servidor a que se destinam estes pedidos. Além destes, pode-se também citar o campo *responseTime*, o qual possui a diferença em segundos do tempo de criação e de tratamento do pedido. Por meio deste, busca-se registrar o tempo aproximado em que esta informação demorou para ser adquirida.

```
{
    "_id": {
        "$oid": "59d76530739ae810b0dcd9cc"
    },
    "checked": true,
    "checkedAt": {
        "$date": "2017-10-06T11:12:48.727Z"
    },
    "createdAt": {
        "$date": "2017-10-06T11:12:48.516Z"
    },
    "requestT": "AutoSimul-03!Kp",
    "responseTime": 0.2116601,
    "responseValue": 1,
    "serverName": "Schneider-Aut.OFS.2"
}
```


Figura 10. Exemplo de documento da coleção *readRequests*.

Alguns campos se diferem entre os dois documentos, como o *responseValue* dos pedidos de leitura, onde é armazenado o valor da resposta, e os campos *responseT* e *requestValue* dos pedidos de escrita. Eles armazenam, respectivamente, uma resposta, se a escrita foi executada ou não, e o valor a ser escrito no dispositivo.

```
{
    "_id": "KjtEh4zgA6xE3ffwb",
    "checked": true,
    "checkedAt": {
        "$date": "2017-10-05T12:19:47.896Z"
    },
    "createdAt": {
        "$date": "2017-10-05T12:19:46.197Z"
    },
    "requestT": "FI302-AO.OUT.VALUE",
    "requestValue": 15,
    "responseT": "Write succeed",
    "responseTime": 1.6994051,
    "serverName": "Smar.hseoleserver.0"
}
```


Figura 11. Exemplo de documento da coleção *writeRequests*.

O banco de dados definido com as características supracitadas será, portanto, capaz de armazenar informações coletadas dos servidores OPC, como os valores lidos dos dispositivos, e também de armazenar estados atuais das conexões do *Cliente OPC* com os servidores. Estas informações poderão ser consultadas pelo cliente da *Web Application*, como será descrito na subseção seguinte.

2.2.3 Web Application

` A *Web Application* foi desenvolvida utilizando-se o *framework Meteor*. Ela será responsável por disponibilizar as informações contidas no banco de dados para o usuário e também permitirá que o mesmo faça algumas requisições aos servidores OPC. O usuário se conectará como um cliente HTTP através de navegadores, como por exemplo o *Google Chrome*, e tendo acesso à Internet. Desta forma, almeja-se que a aplicação dê ao usuário os seguintes direitos:

- Saber informações dos servidores OPC registrados pelo *Cliente OPC*, como por exemplo: seu nome identificador, se o mesmo está *online*, a data em que o mesmo o esteve pela última vez e se está sendo realizada uma aquisição de dados periódica no mesmo.

- Vizualizar os dispositivos registrados para cada servidor OPC, bem como realizar pedidos de leitura ou escrita nestes dispositivos. O último valor e a data de aquisição do mesmo, para cada dispositivo, também poderá ser visto.

- Vizualizar os detalhes de cada um dos 10 últimos pedidos de leitura e de escrita realizados, como: se já foi tratado pelo servidor, datas de criação e de tratamento, além de um tempo aproximado entre estes dois últimos eventos citados;

Nota-se que as informações que serão disponibilizadas para o cliente podem ser associadas diretamente aos documentos contidos no banco de dados e aos seus campos chave-valor. Adicionalmente, tem-se que os componentes que aparecem na tela para o usuário são definidos no código em HTML da aplicação, mais especificamente dentro da *tag body* a qual representa o corpo principal da página. Desta forma, foram utilizados mecanismos para modificar o conteúdo do código contido no *body* em função dos documentos existentes no banco de dados. Isso possibilitará que o usuário da aplicação veja as mudanças feitas no banco de dados pelo *Cliente OPC*.

Da biblioteca em *JavaScript* do *Meteor,* foi importada a classe *Template*, a qual criará um objeto para cada *template* em HTML definido no programa, assim como um objeto é criado para o *body*. Por sua vez, estes objetos possuem o atributo *helpers*, onde foi adicionado, como um dicionário chave-valor, as funções que serão responsáveis por buscar os documentos no banco de dados. Estas funções retornam os documentos em um *array*, os quais são manipulados e inseridos no *body* por meio da biblioteca do *Meteor* chamada *Blaze*.

Na **Figura 12**, podem ser vistas algumas das funções definidas como *helpers* para retornarem objetos que serão utilizados no *body*.

```
Template.body.helpers({
  serversOPC(){                 //retorna todos os documentos que representam os servidores OPC registrados no banco de dados
    return ServersOPC.find({});
  },
  readRequestsSubmited(){       //Retorna os documentos das 10 últimas requests de leitura
    return ReadRequests.find({},{ sort: { createdAt: -1 }, limit:10  });
  },
  writeRequestsSubmited(){      //Retorna os documentos das 10 últimas requests de escrita
    return WriteRequests.find({},{ sort: { createdAt: -1 }, limit:10 });
  },
  devicesAvailableInServer(serverN){//Retorna os documentos que representam os dispositivos de um servidor OPC
    return DevicesAvailable.find({serverName: serverN});
  },
```

Figura 12. Funções de busca no banco de dados: definidas como *helpers* do *body*.

Os nomes das funções *serversOPC*, *readRequestsSubmitted*, *writeRequestsSubmitted* e *devicesAvailableInServer* foram utilizados no código em HTML do *body* juntamente com as diretivas do *Blaze*. Como exemplo, a função *serversOPC* retorna em um *array* todos os documentos contidos na coleção *serversOPC*. Os elementos deste *array*, como os campos dos documentos, também podem ser inseridos em HTML. Para inserir o nome do servidor que está contido no campo *serverName*, basta escrever *{{serverName}}*. Na **Figura 13**, este exemplo de inserção pode ser conferido.

```
<ul class="dropdown-menu">
  {{#each serversOPC}}
    <li role="presentation" class="text-center">
      <a href="#tab-{{idParserTabControl _id}}" aria-controls="tab-{{idParserTabControl _id}}" role="tab" data-toggle="tab">
              <h1><span class="label label-{{#if connected}}success{{else}}default{{/if}}"> {{serverName}}</span></h1>
      </a>
    </li>
  {{/each}}

</ul>
```

Figura 13. Utilização das diretivas do *Blaze* para colocar os nomes dos servidores em um menu *dropdown*.

Utilizando a diretiva *{{#each serverOPC}}*, o código contido dentro desta diretiva será repetido para cada elemento do *array* retornado pela função, como também visto na **Figura 13**. No exemplo apresentado, objetiva-se disponibilizar, em um menu do estilo *dropdown*, os nomes dos servidores OPC registrados no banco de dados e defini-los como *links* em que o usuário possa clicar. Na **Figura 14**, pode-se ver o efeito destes mecanismos quando se tem no banco de dados três documentos na coleção s*erversOPC* com os nomes de *"Smar.hseoleserver.0"*, *"Advosol.DA3CBCS.1"* e *"Schneider-Aut.OFS.2"*.

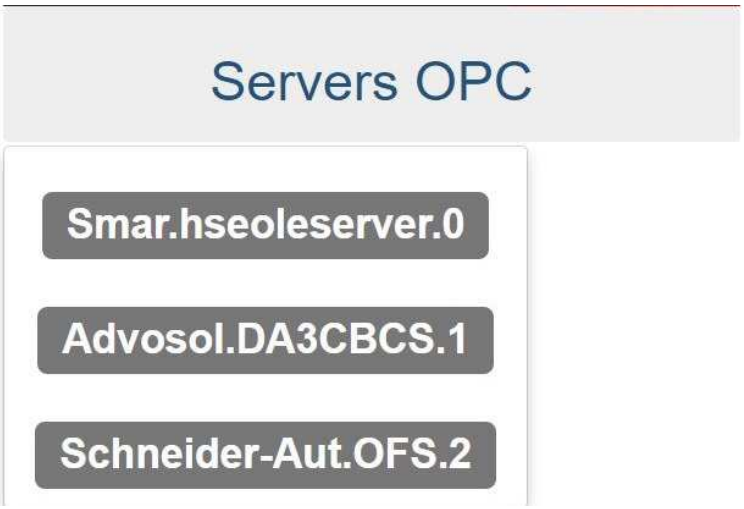


Figura 14. Menu *dropdown* criado em HTML com o *helper serversOPC* e diretivas *Blaze*.

De forma semelhante à descrita anteriormente, por meio da função *devicesAvailableInServer*, serão buscados na coleção *devicesAvailable* os documentos que representam os dispositivos contidos em um dado servidor OPC, sendo seu nome passado como parâmetro de entrada da função. Para cada dispositivo encontrado, será criado um painel que terá um botão e uma caixa de texto para que possam ser feitas requisições de leitura e escrita pelo usuário. Além destes controles, existirão espaços reservados para se mostrar o valor e a data de aquisição do último pedido de leitura requisitado ao dispositivo em questão. Estes dados também são buscados através de funções como anteriormente citado. Na **Figura 15**, são mostrados os painéis para quando se tem dois dispositivos com seus identificadores *"TT302-AI.OUT.VALUE"* e *"FI302-AO.OUT.VALUE"*.

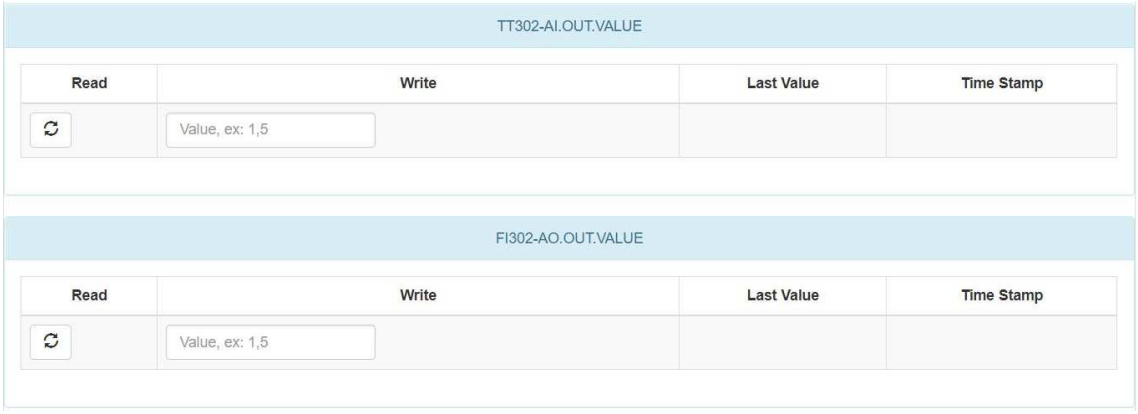


Figura 15. Paineis criados com o *helper devicesAvailableInServer* e as diretivas *Blaze*.

Já as funções *readRequestsSubmitted* e *writeRequestsSubmitted* foram utilizadas para se buscar, respectivamente, nas coleções *readRequests* e *writeRequests* do banco de dados, os últimos documentos inseridos em cada coleção. A quantidade de documentos é limitada em no máximo dez, sendo que estes documentos serão representados como linhas de uma tabela, conforme o exemplo da **Figura 16**, que mostra três pedidos de leitura.

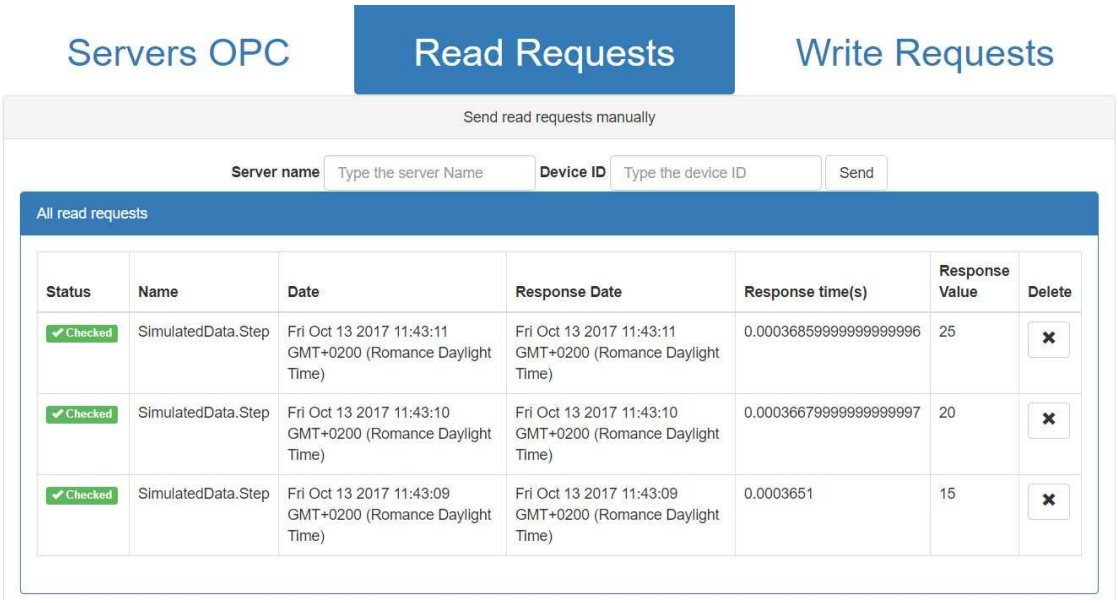


Figura 16. Linhas da tabela criadas com o *helper readRequestsSubmitted* e diretivas *Blaze*.

Para este caso foram feitos templates HTML para os pedidos de leitura e escrita, os quais são simplesmente a linha de uma tabela onde são colocados, nas suas devidas colunas, os campos dos documentos que acessam os valores da referida coluna. Os *templates* podem ser vistos na **Figura 17** e na **Figura 18**.

```
1  <template name="TempReadRequest">
2    <tr>
3      <td><span class="label label-{{#if checked}}success{{else}}default{{
       /if}}">{{#if checked}}<span class="glyphicon glyphicon-ok"
       aria-hidden="true"></span> Checked{{else}}Waiting Server...{{/if}}</
       span></td>
4      <td> {{requestT}}</td>
5      <td> {{createdAt}}</td>
6      <td> {{checkedAt}} </td>
7      <td> {{responseTime}}</td>
8      <td> {{responseValue}}</td>
9      <td><button class="btn btn-default delete-ReadRequest"> <span class=
       "glyphicon glyphicon-remove" aria-hidden="true"></span> </button></
       td>
10   </tr>
11 </template>
```

Figura 17. *Template* para os documentos de pedidos de leitura.

```
<template name="TempWriteRequest">
  <tr>
    <td><span class="label label-{{#if checked}}success{{else}}default{{
    /if}}">{{#if checked}}<span class="glyphicon glyphicon-ok"
    aria-hidden="true"></span> Checked{{else}}Waiting Server...{{/if}}</
    span></td>
    <td> {{requestT}}</td>
    <td> {{createdAt}}</td>
    <td> {{checkedAt}}</td>
    <td> {{responseTime}}</td>
    <td> {{requestValue}}</td>
    <td> {{responseT}}</td>
    <td><button class="btn btn-default delete-WriteRequest"> <span class
    ="glyphicon glyphicon-remove" aria-hidden="true"></span> </button></
    td>
  </tr>
</template>
```

Figura 18. *Template* para os documentos de pedidos de escrita.

Os *templates* também são inseridos com as diretivas do *Blaze*, porém utilizando um sinal de "**>**" e, a seguir, o nome do *template*, como pode ser visto no exemplo da **Figura 19**.

```
<div class="panel panel-primary">
  <div class="panel-heading">All read requests</div>
  <div class="panel-body">
    <div class="table-responsive">
     <table name="tabelaReadRequest" class="table table-bordered table-hover">
        <thead>
          <tr>
            <th>Status</th>
            <th>Name</th>
            <th>Date</th>
            <th>Response Date</th>
            <th>Response time(s)</th>
            <th>Response Value</th>
            <th>Delete</th>
          </tr>
        </thead>
        <tbody>
          {{#each readRequestsSubmited}}
            {{> TempReadRequest}}
          {{/each}}

        </tbody>
      </table>
    </div>

  </div>
</div>
```

Figura 19. Utilização do *helper readRequestsSubmitted* e a inserção do *template* pelas diretivas *Blaze*.

Um outro atributo pertencente aos objetos criados para os *templates* e o *body*, assim como os *helpers*, é chamado de *events*. Como o nome sugere, nele são definidos os eventos e suas respectivas funções de tratamento. Na **Figura 20**, podem ser vistas as definições das

funções de tratamento do clique no botão para submissão do pedido de leitura e para submissão do formulário para pedido de escrita no dispositivo.

```
Template.body.events({
  'click .device-ReadRequest'(event) {//clique no botão de leitura do dispositivo
    // Chamando método para inserir um novo pedido de leitura no banco de dados
    Meteor.call('readRequests.insertReadRequests',this.serverName,this.deviceID)
  },
  'submit .deviceNew-writeRequest'(event) {//formulário é submetido
    event.preventDefault();
    const target = event.target; // Pegando valor do elemento
    const valueRequest = target.devWriteRequestServer.value;
    // Chamando método para inserir um novo pedido de escrita no banco de dados
    Meteor.call('writeRequests.insertWriteRequests',this.serverName,this.deviceID,valueRequest)
    // Clear form
    target.devWriteRequestServer.value = '';
  },
```

Figura 20. Funções de tratamento de eventos: definidas como *events* no *body*.

Após o clique no botão que possui um atributo *"device-ReadRequest"*, um método será chamado para que seja criado um novo pedido de leitura endereçado ao servidor OPC e dispositivo em questão. Por sua vez, ao se submeter o formulário que possui um atributo de nome *"deviceNew-writeRequest"*, o valor digitado será capturado e será, então, chamado um método para criar um novo pedido de escrita. O pedido também será feito ao servidor e dispositivos em questão, levando consigo o valor digitado pelo usuário.

Os eventos que serão gerados pelo usuário e as informações contidas no banco de dados serão gerenciados e disponibilizadas na tela do cliente como descrito acima.

Com a biblioteca *Bootstrap*, foi criado o *design* dos componentes, assim como sua adaptação aos diferentes tamanhos de tela dado o dispositivo em que se acessa a aplicação. A biblioteca é facilmente adicionada no projeto *Meteor* com o comando: *"meteor add twbs:bootstrap"*, o qual pode ser consultado no site **[21]**, que possui diversas bibliotecas que podem ser utilizadas com o *Meteor*. A partir deste momento, consultando-se a referência da *Bootstrap*, foram adicionados os atributos às *tags* HTML do *body* e o *design* da aplicação foi definido. Na **Figura 21**, pode-se ver a adaptação das dimensões dos componentes a um celular, o que é uma das vantagens providas pela biblioteca utilizada.

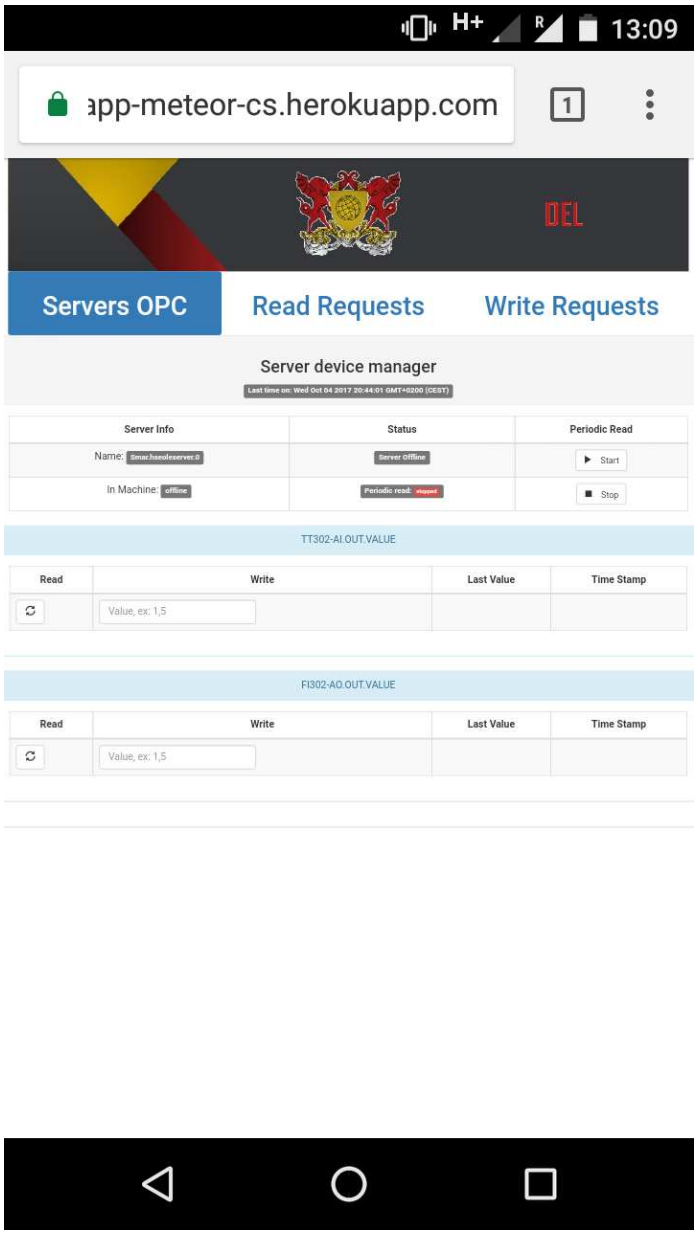


Figura 21. Aplicação sendo acessada através de um navegador em um celular.

Após a descrição das três partes que compõem o sistema, no capítulo seguinte serão feitas as discussões e a apresentação dos resultados obtidos.

3 Resultados e Discussões

Neste capítulo, será apresentado o funcionamento do sistema e dois exemplos de aplicação do mesmo. Na seção 3.1, serão demonstradas suas funcionalidades e o que os usuários de cada aplicação podem fazer com as mesmas. No exemplo da seção 3.2, foram realizadas operações de leitura e escrita em dispositivos de uma rede *Fieldbus*, sendo estes configurados em um servidor OPC fornecido pela empresa *SMAR*. Já na seção 3.3, variáveis de um controlador PID, o qual era realizado por um CLP para controlar um pequeno motor de corrente contínua, foram lidas e modificadas. No último exemplo, foi utilizado um servidor OPC da *Schneider Electric*.

Nos testes, os servidores OPC e o *Cliente OPC* foram executados na mesma máquina, tendo sido utilizadas máquinas virtuais. O *software* utilizado para criação das máquinas virtuais foi o *Oracle VM VirtualBox 5.1.26*. O servidor da *Web Application* foi hospedado com auxílio da plataforma *Heroku* **[22]**, a qual disponibiliza serviços de hospedagem para aplicações do tipo *Node.js*, como a que foi criada neste trabalho com o *Meteor*. O acesso ao site pode ser feito pela URL: https://webapp-meteor-cs.herokuapp.com/.

3.1 Demonstração do sistema

Para realizar a demonstração do sistema foi utilizado o servidor de testes da *Advosol*, o qual possui o nome *"Advosol.DA3CBCS.1"* e é automaticamente instalado com o pacote de desenvolvimento de clientes citado na seção 1. Este servidor foi executado em uma máquina virtual com sistema operacional *Windows 7 – 64 bits* como *guest* e *Windows 10 – 64 bits* como host, a qual possuía acesso à internet.

Uma instância do *Cliente OPC* foi executada e, assim que a interface foi disposta na tela, foram realizadas as seguintes ações no grupo de controles *"Server Connection"*:

- Um clique no botão *"Browse Servers"* para que fossem mostradas no *ComboBox* as *strings* identificadoras dos servidores OPC existentes na máquina;
- O nome *"Advosol.DA3CBCS.1"* foi selecionado na *ComboBox*;

- Um clique no botão *"Connect"* para iniciar a conexão com o servidor.

Na **Figura 22** pode ser visto o estado da interface após estas ações, podendo-se conferir mais detalhes no *Status log.*

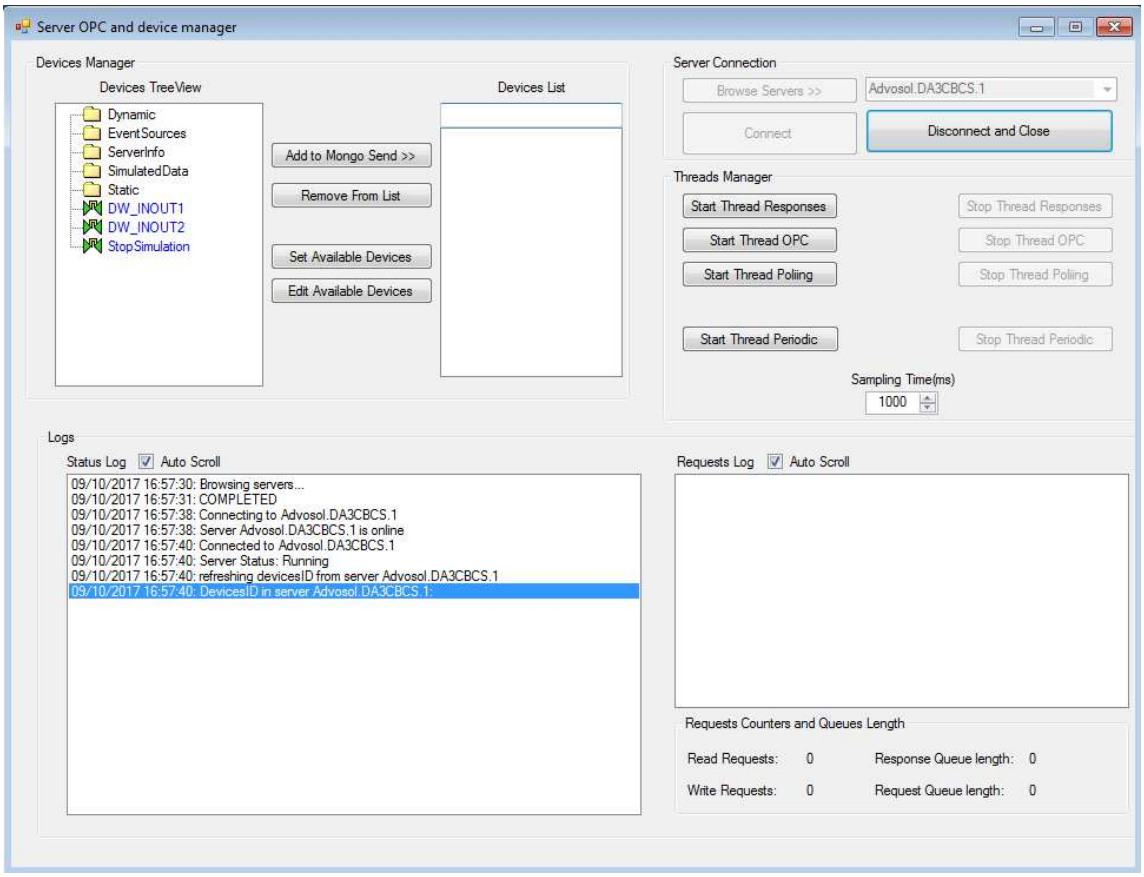


Figura 22. Estado da interface após busca pelos servidores OPC e conexão com o servidor de testes da *Advosol*.

O site da *Web Application* foi acessado por um navegador no mesmo computador, como pode ser visto na **Figura 23**. Na tela pôde-se ver que o nome do servidor foi adicionado entre as opções de servidores OPC no menu do tipo *drop down*. O nome do mesmo está sinalizado em verde, o que significa que há uma conexão com o mesmo.

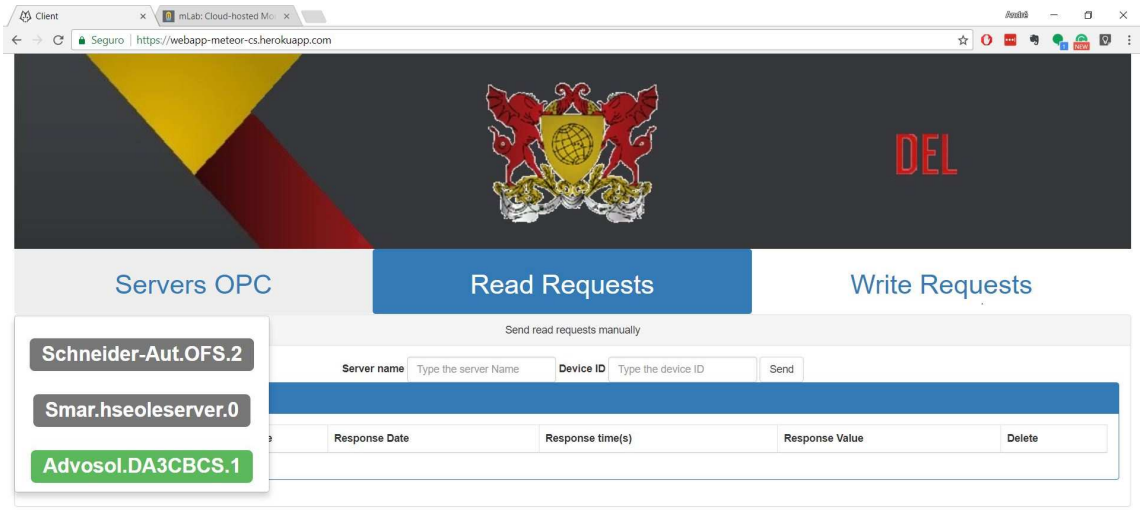


Figura 23. Acesso da aplicação e verificação da conexão com o servidor "Advosol.DA3CBCS.1".

Foi clicado no nome do servidor e na tela da aplicação foi disposto um painel com mais informações acerca do mesmo. Entre tais informações estão o horário de conexão e o nome da máquina em que está executando, como pode ser visto na **Figura 24**.

Figura 24. Acesso às informações atuais do servidor após clique no menu *dropdown*.

Retornou-se à interface do *Cliente OPC* e as ações que se seguem foram feitas nos controles do grupo *Devices Manager* e *Threads Manager:*

- Através da *TreeView*, navegou-se pelos dispositivos disponíveis no servidor e o dispositivo *"Step"* foi selecionado. Este dispositivo possui um valor simulado por um incremento de 5 em 5 entre 0 e 100, reiniciando-se em 0 quando atinge o valor máximo de 100;

- Um clique no botão *"Add to Mongo Send"*. Desta forma foi colocada a *string* identificadora do dispositivo na lista de dispositivos selecionados e no *ComboBox* localizado à direita do botão;

- Um clique no botão *"Set Available Devices"*, para registrar os dipostivos da lista no banco de dados.

- Um clique em cada um dos botões *"Start Thread Responses"*, *"Start Thread OPC" e "Start Thread Polling"*, inicializando, portanto, as atividades do *software*.

Na **Figura 25** pode ser visto, pelo *Status log*, a adição do dispositivo no banco de dados, assim como a confirmação de início das *threads*.

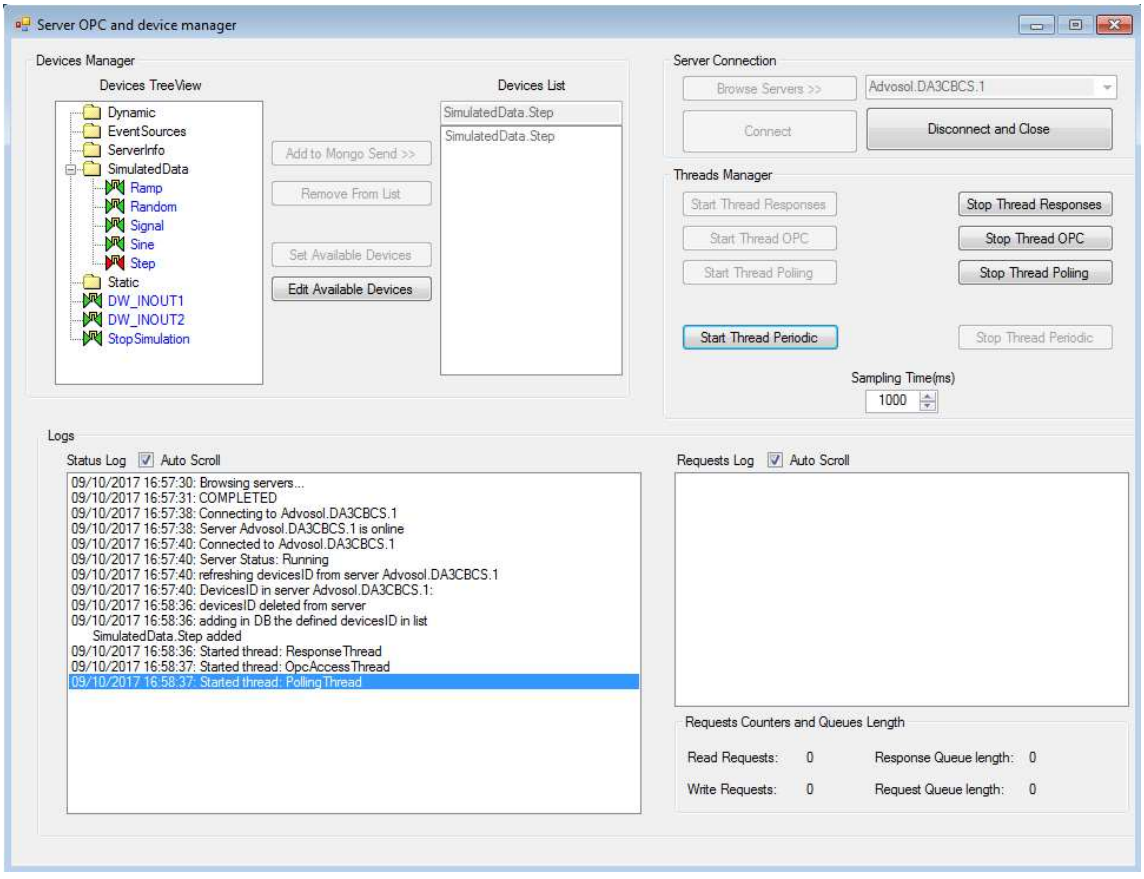


Figura 25. Estado da interface após adição do dispositivo no banco de dados e início das *threads*.

O registro do dispositivo no banco de dados pôde também ser conferido pelo *site*, porque um novo painel foi adicionado na interface do mesmo. Por meio deste painel, intitulado com o nome indentificador do dispositivo, o usuário foi autorizado a fazer pedidos

de leitura e escrita no mesmo. O painel adicionado, assim como seus controles, pode ser visto na **Figura 26**.

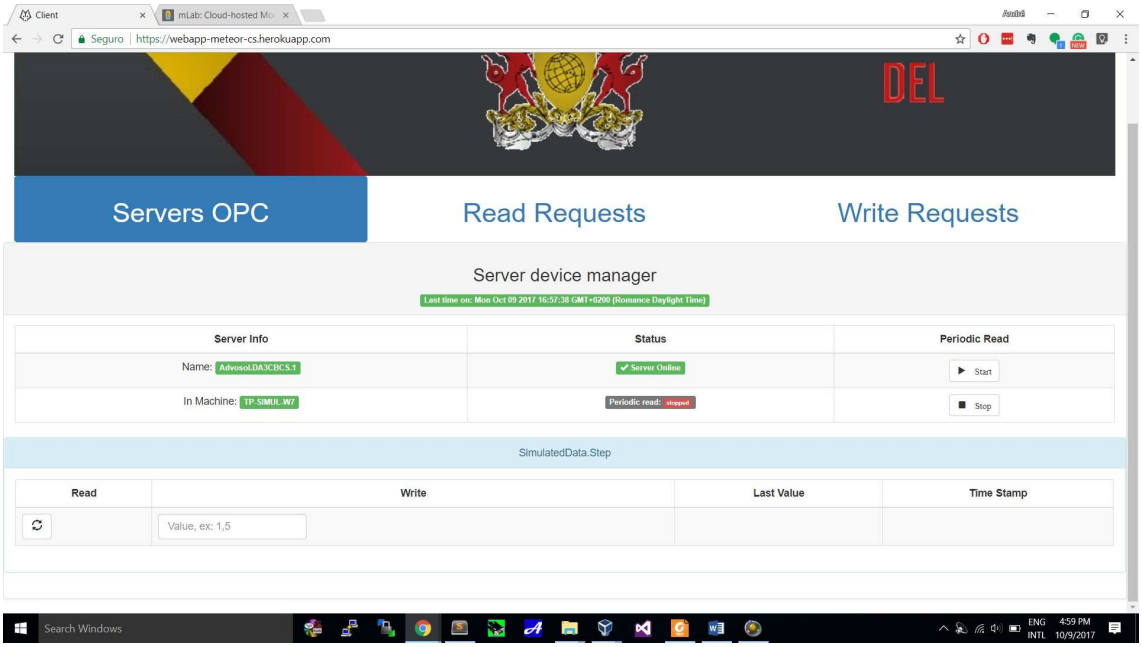


Figura 26. Disposição de um painel para o dispositivo após o mesmo ser adicionado ao banco de dados.

Foi feito um pedido de leitura, clicando-se no botão com símbolo indicando atualização que se encontra logo abaixo do título *"Read"*. Na **Figura 27**, analisando-se o *Status Log* e o *Requests Log*, pode ser visto o momento de identificação do pedido e também o valor lido do servidor OPC, inclusive com seus respectivos horários. No *Requests Log*, os horários das leituras são gravados em *Coordinated Universal Time* (UTC), enquanto que os horários dos eventos mostrados no *Status Log* e no site estão no fuso local do teste. Isto explica a diferença de duas horas entre os valores, dado que o teste foi feito em um local com duas horas a mais do que o UTC.

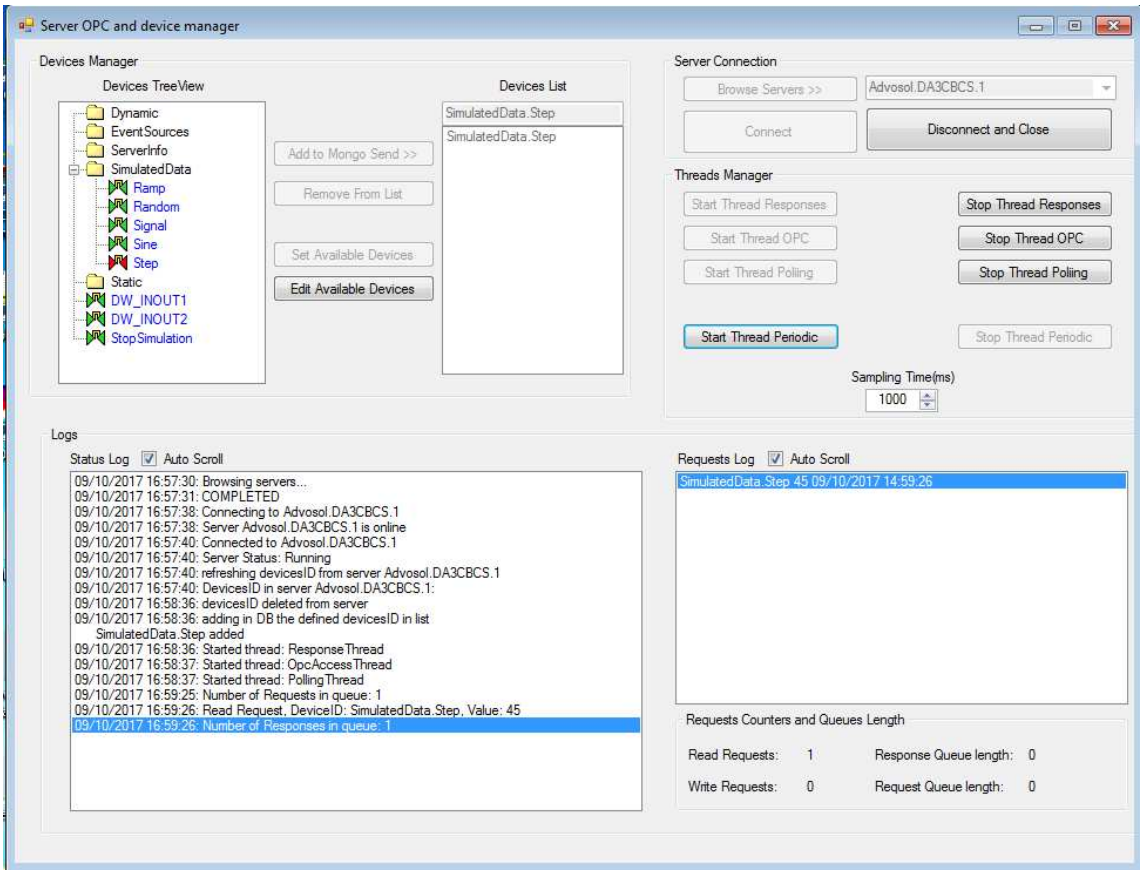


Figura 27. Estado da interface após identificação do pedido de leitura e resposta do mesmo pelo servidor **OPC.**

Já na **Figura 28**, pode ser visto que o valor e seu respectivo horário de aquisição, no fuso do cliente, foram mostrados no painel do dispositivo em questão.

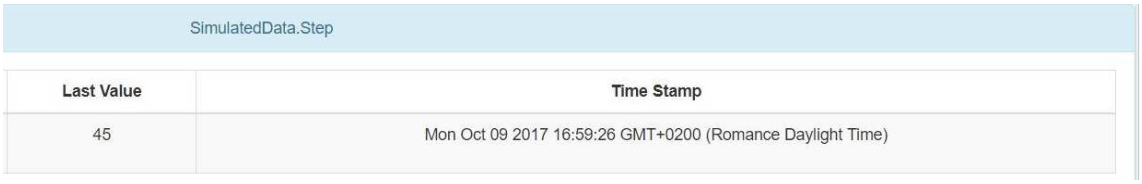


Figura 28. Disposição no painel do valor e data de resposta da requisição gerada.

Em sequência, foi feito um pedido de escrita do valor 1 no dispositivo e, logo em seguida, um pedido de leitura no mesmo. A requisição de escrita foi feita digitando-se o número 1 na caixa de texto abaixo do título *"Write"* e apertando-se *"Enter"*. A requisição de leitura foi feita como anteriormente explicado. Na **Figura 29**, podem ser verificados estes pedidos sendo identificados e tratados pelo *Cliente OPC*, conforme destacado no *Status log.*

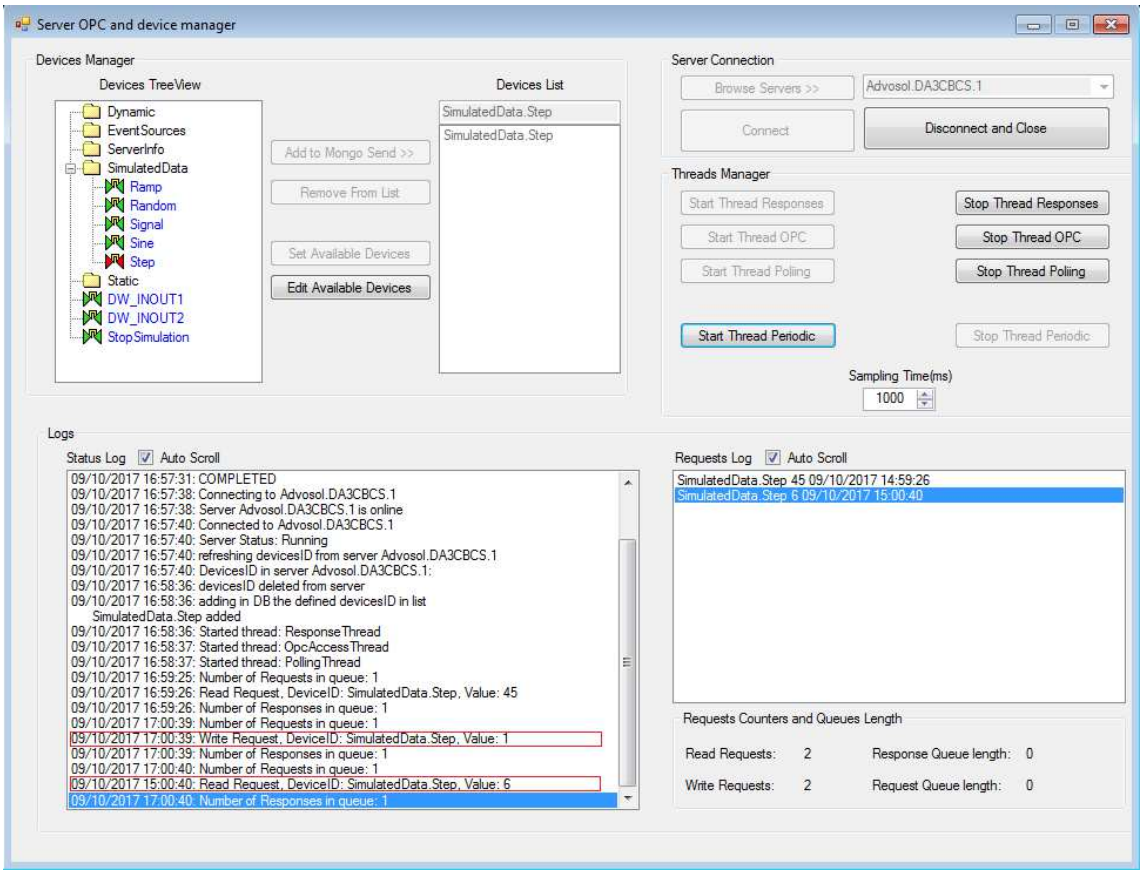


Figura 29. Estado da interface após identificação dos pedidos realizados e a resposta do servidor OPC.

A atualização do novo valor lido pode ser conferida na **Figura 30**. Nota-se que o valor lido é 6, não tendo sido o pedido de leitura e seu respectivo processamento rápidos o suficiente para adquirir o valor escrito que foi 1. Ou seja, já havia ocorrido uma mudança do valor, a qual acontece acrescentando-se 5 no valor a cada 500 milisegundos. Porém, pode-se concluir que a escrita aconteceu, uma vez que não seria possível captar o valor 6 se ela tivesse falhado.

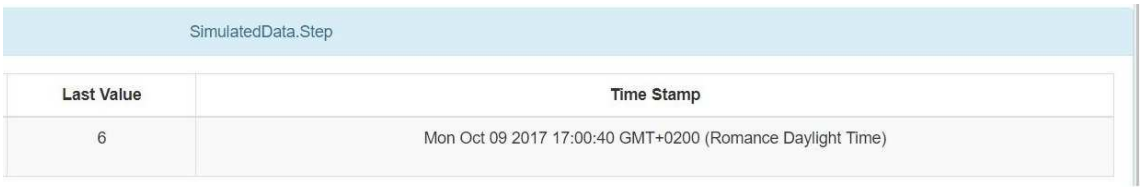


Figura 30. Atualização no painel do valor e data de resposta da requição de leitura feita.

Os detalhes das requisições realizadas podem ser vistos nas abas *"Read Requests"* e *"Write Requests"* do site, conforme ilustrado, respectivamente, na **Figura 31** e na **Figura 32**.

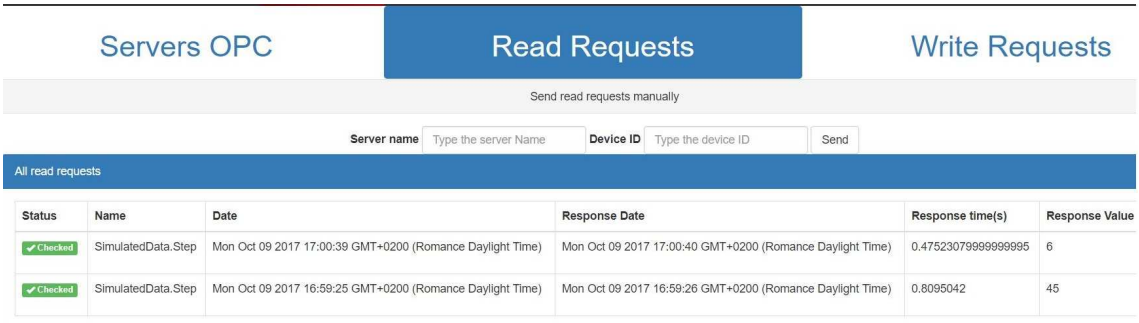


Figura 31. Aba *Read Requests*: detalhes das duas requisições de leitura feitas.

Os pedidos de leitura tiveram tempo de aproximadamente 0.475 e 0.81 segundos, sendo este tempo entre a requisição feita pelo cliente através do site e a recepção da resposta do servidor OPC pelo *Cliente OPC*. O pedido de escrita teve um tempo aproximado de 1.35 segundos.

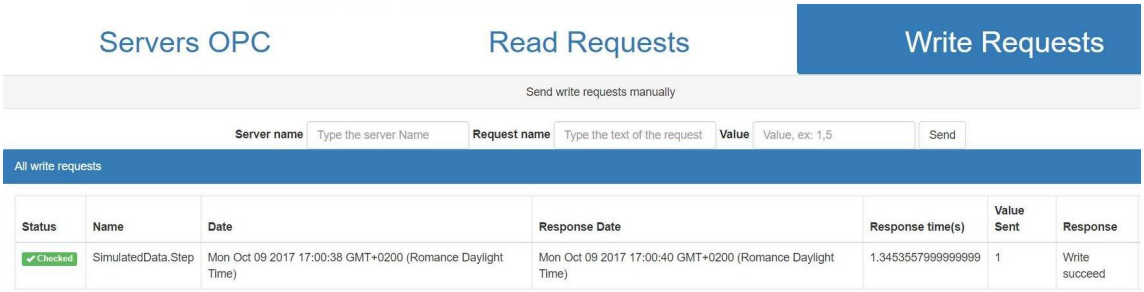


Figura 32. Aba *Write Requests*: detalhes da requisição de escrita realizada.

Por fim, clicou-se no botão *"Disconnect and Close"* do grupo de controles *"Server Connection"*. Desta forma, a conexão com o servidor foi encerrada, os dados dos *Logs* foram salvos e a aplicação finalizada. Os dados contidos no *Status Log* foram salvos em um arquivo *.txt* que possui o nome *"statusLog-2017-10-9--StartedAt-16h-57m-29s"*, contendo a data de início de execução da aplicação. Por sua vez, os dados contidos no *Requests Log* foram salvos em arquivos estruturados nos formatos *.txt* e *.csv*, ambos contendo o nome *"requestsLog-2017-10-9--StartedAt-16h-57m-29s"*.

No *site* pôde ser visto que a conexão foi finalizada, conforme a **Figura 33**. Nas informações sobre o servidor *"Advosol.DA3CBCS.1"* existe a indicação do estado *offline*, bem como a mudança da cor verde para cinza.

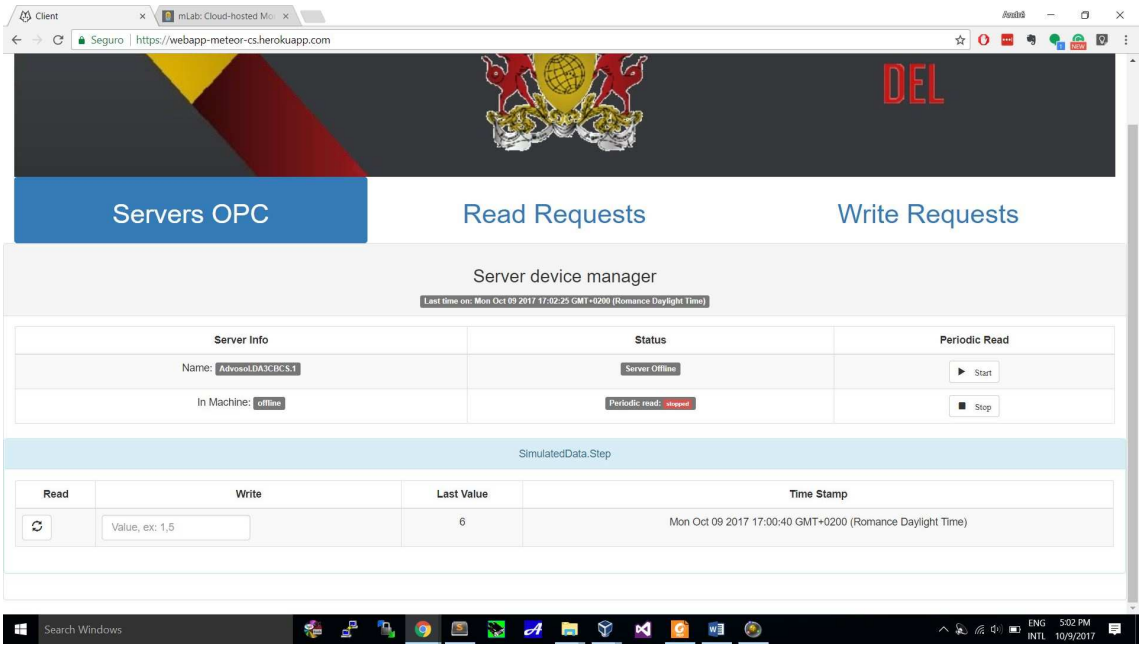


Figura 33. Estado da tela após desconexão do servidor OPC.

Além das ações exemplificadas anteriormente, existe a possibilidade da leitura periódica de todos os dispositivos registrados de um dado servidor OPC. Quando este processo é ativado, o *Cliente OPC* se encarrega de requisitar leituras periodicamente nos dispositivos, sem que seja necessário que o usuário as faça manualmente. Tal procedimento pode ser iniciado ou parado tanto pela *Web Application* quanto pelo *Cliente OPC*, tal qual será mostrado no teste a seguir.

Uma nova instância da aplicação foi lançada e foram feitas todas as etapas de conexão e início das *threads.* O dispositivo anteriormente registrado, como esperado, aparece na *ComboBox*. Isso ocorre porque logo após o início da conexão com o servidor OPC são buscados todos os dispositivos já registrados para aquele servidor. A seguir, foi feito um clique no botão *"Start Thread Periodic"* e, alguns segundos depois, um clique no botão *"Stop Thread Periodic"*. Os resultados destas ações podem ser verificados no *Status Log* por meio da **Figura 34**, onde se destaca os momentos de início e parada da *thread* de leitura periódica. Foram executados três pedidos de leitura no dispositivo registrado.

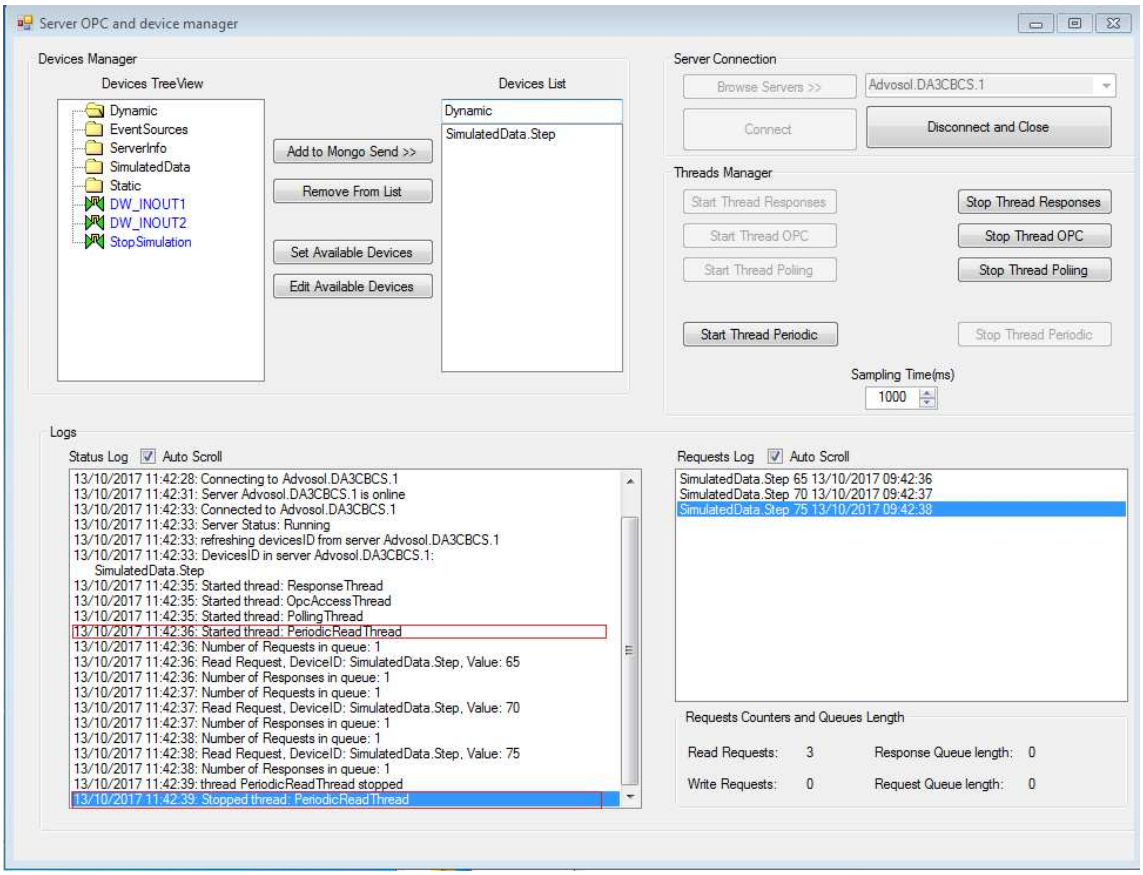


Figura 34. Estado da interface após pedidos de início e parada da leitura periódica pelo Cliente OPC.

Acessando-se novamente o site, como descrito anteriormente, clicou-se no botão *"Start"* e, novamente, alguns segundos depois clicou-se no botão *"Stop"*, ambos localizados abaixo do título *"Periodic Read"*. Um resultado semelhante ao visto na **Figura 34**, pode ser visto na **Figura 35**. A diferença existente é o registro dos momentos de identificação dos pedidos de início e parada, vindos da *Web Application.* Foram feitos mais quatro pedidos de leitura no dispositivo, totalizando sete pedidos como pode ser visto no contador com nome *"Read Requests"*.

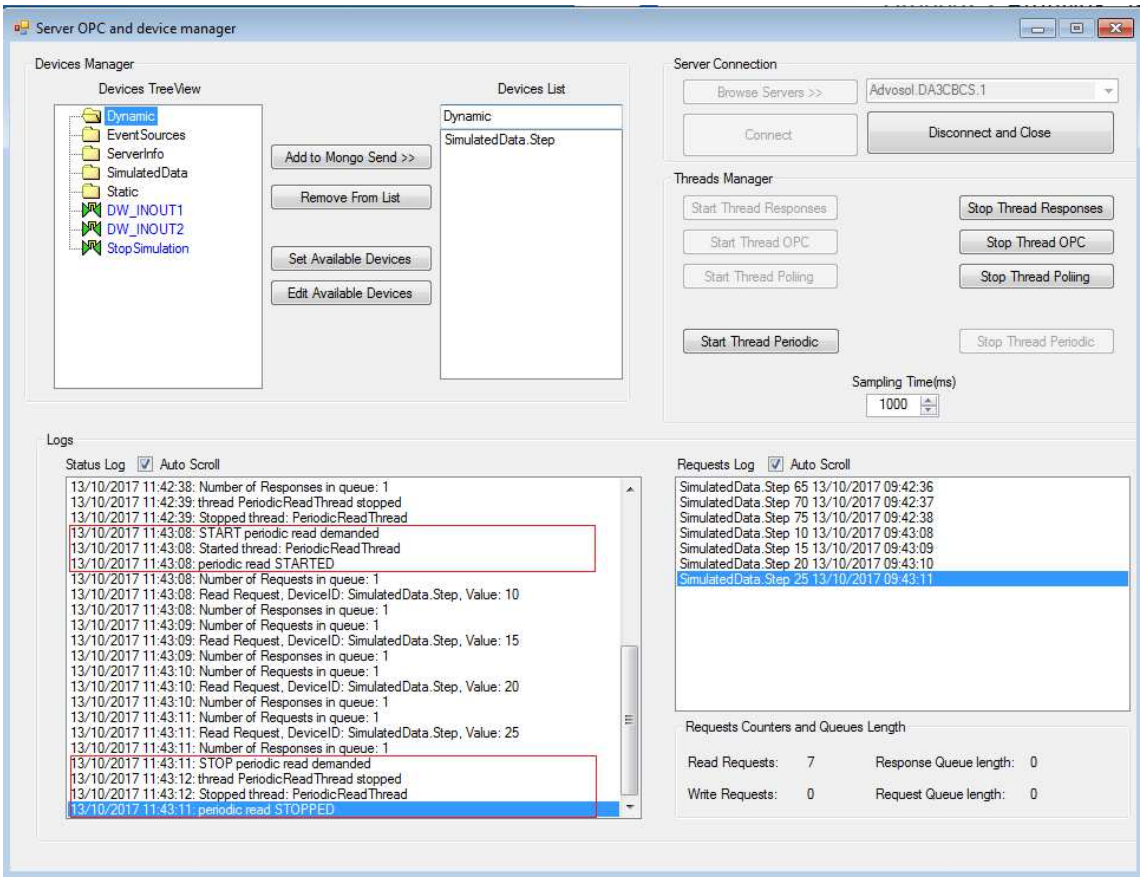


Figura 35. Estado da interface após pedidos de início e parada da leitura periódica pela *Web Application.*

Utilizando-se as possibilidades descritas nesta seção, serão mostrados dois exemplos de aplicação do sistema desenvolvido.

3.2 Exemplo: servidor OPC da Smar

Para o teste em questão, foi utilizado um servidor OPC do fabricante *Smar*. Este servidor foi configurado por meio do *software* fornecido pela empresa chamado *System302* e possui nome identificador *"Smar.hseoleserver.0"* **[23].**

Por meio deste servidor, foi possível acessar os seguintes dispositivos: um *FieldBus Universal Bridge DFI302*, um conversor *FieldBus* para corrente *FI302* e um transmissor de temperatura *Fieldbus TT302*. Estes dispositivos são produzidos pela *Smar* e esta planta se encontra no departamento de engenharia elétrica da UFV, podendo ser vista na **Figura 36 [24][25][26].**

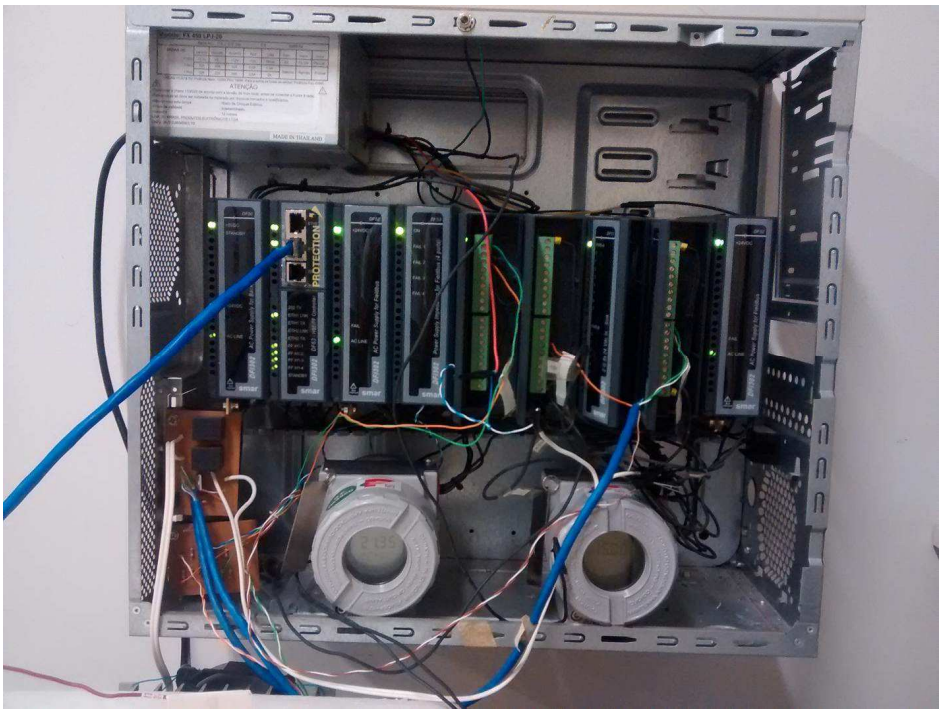


Figura 36. Planta utilizada para no teste desta seção.

Este servidor foi executado em condições similares às utilizadas no teste da seção anterior: em uma máquina virtual com sistema operacional *Windows 7 – 64 bits* como *guest* e *Windows 10 – 64 bits* como host. Em adicional, a máquina virtual estava conectada na rede local do laboratório possuindo um IP definido e acesso à internet.

Uma instância do *Cliente OPC* foi lançada na máquina virtual e foi realizada a conexão com o servidor. Por meio da *TreeView*, os identificadores dos valores de saída dos dispositivos *TT302* e *FI302*, sendo estes *"TT302-AI.OUT.VALUE"* e *"FI302-AO.OUT.VALUE"*, foram registrados no banco de dados. Na **Figura 37**, podem ser conferidas, à esqueda no *Status Log* gerado na interface gráfica, as ações realizadas. À direita é mostrada a localização em que este sistema se encontra: no departamento de engenharia elétrica da UFV.

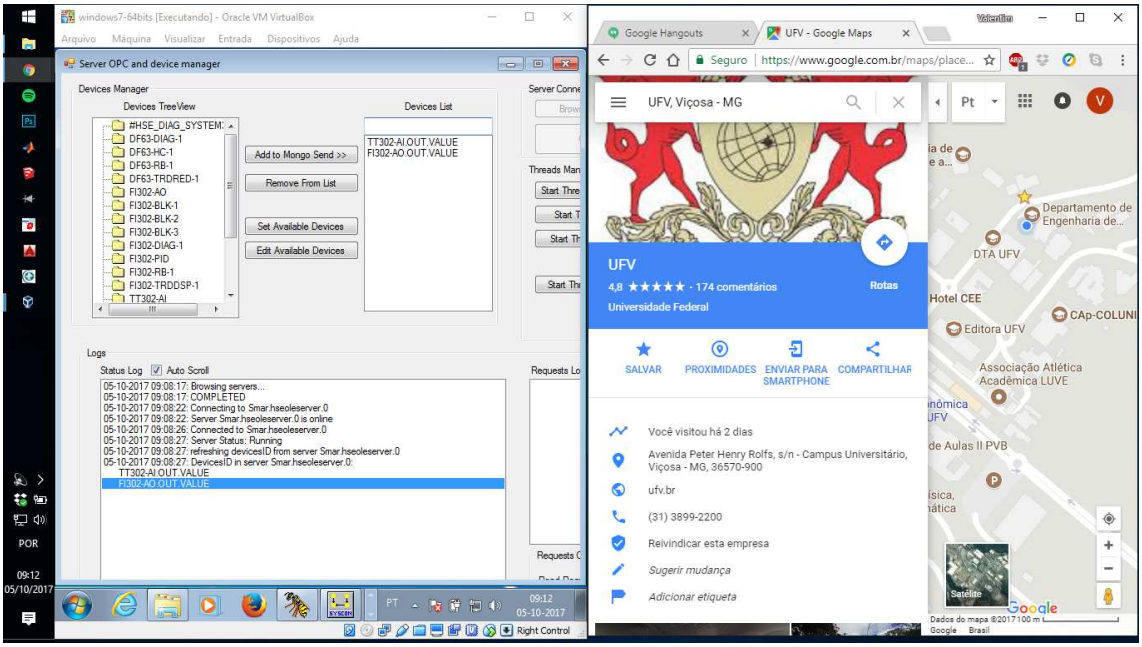


Figura 37. Registro no banco de dados dos dispositivos e localização da máquina onde o Cliente OPC é executado.

Por um computador localizado na *École Nationale Supérieure d'Eléctricité et de Mécanique* (ENSEM) e utilizando o navegador *Google Chrome*, o site foi acessado. Foi possível, então, verificar que uma conexão foi feita no servidor *"Smar.hseoleserver.0"*. Estas ações podem ser verificadas pela **Figura 38**.

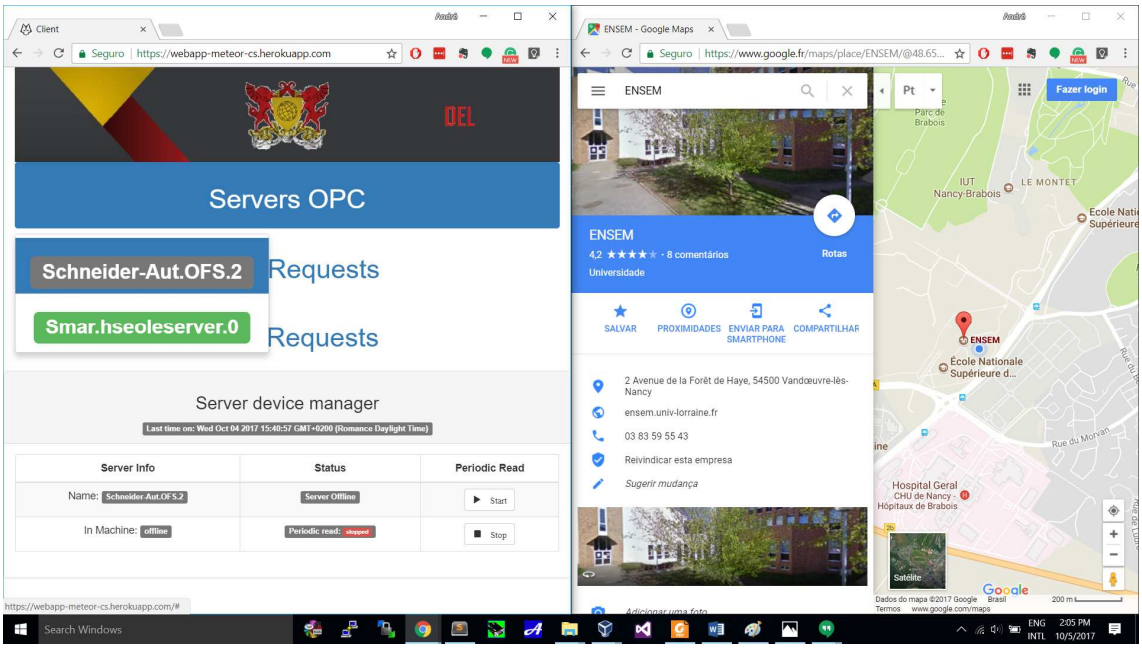


Figura 38. Identificação da conexão com o servidor pela *Web Application* e localização de onde é feito o acesso.

Através do site foram realizados pedidos de escrita e leitura no dispositivo *"FI302-AO.OUT.VALUE"*, sendo primeiramente escrito o valor 10 e, em seguida, o valor 15. Estas requisições foram executadas pelo *Cliente OPC* conforme pode ser verificado no arquivo *.txt* que contém o *Status Log* desta conexão, ilustrado na **Figura 39**.

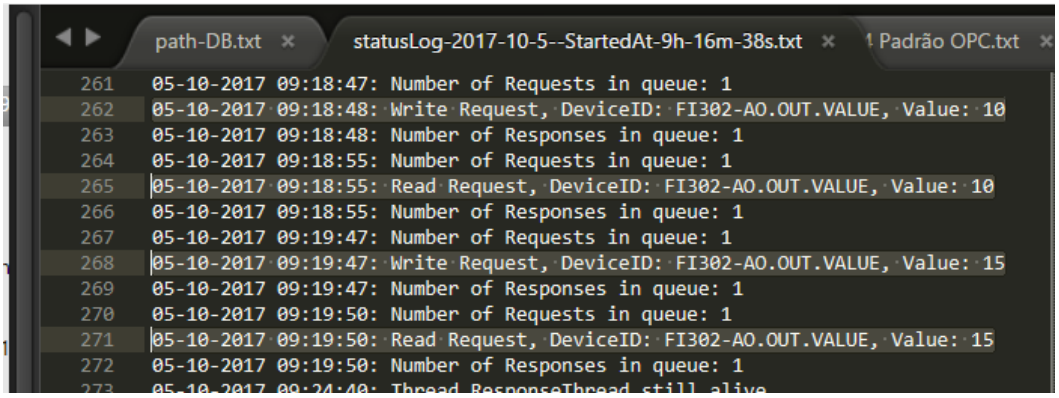


Figura 39. Arquivo .txt gerado: identificação das requisições feitas.

A atualização destes valores de leitura pôde ser monitorada pelo site, conforme visto na **Figura 40**. Na parte esquerda da figura tem-se o estado após a escrita do valor 10 no dispositivo e um pedido de leitura no mesmo. Por sua vez, na parte direita da figura tem-se o estado após a escrita do valor 15 e um outro pedido de leitura. Os horários das aquisições no

Status Log e no site diferem somente em razão da diferença de fuso horário de 5 horas existente entre a França e o Brasil no momento do teste.

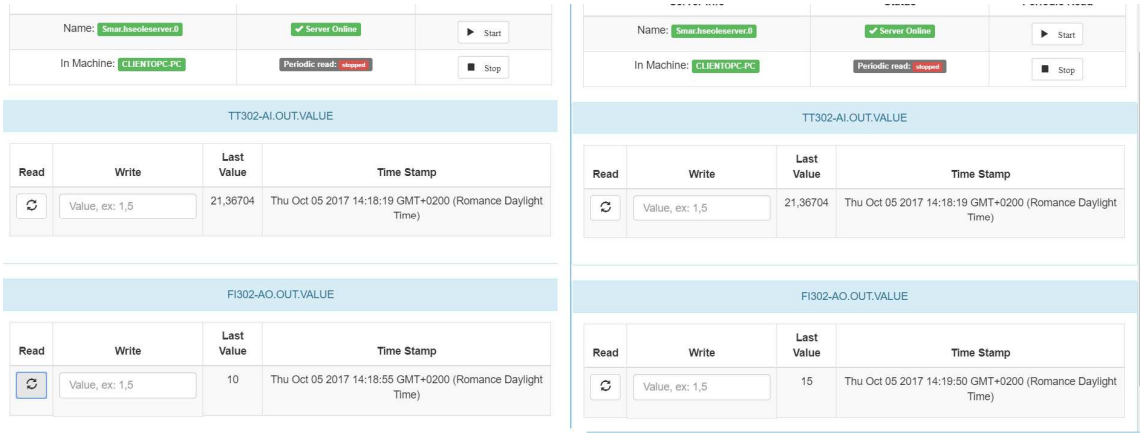


Figura 40. Estados da tela após a execução dos pedidos de escrita e leitura.

Foram retiradas fotos dos dispositivos registrando os dois estados supracitados, as quais podem ser conferidas na **Figura 41**.

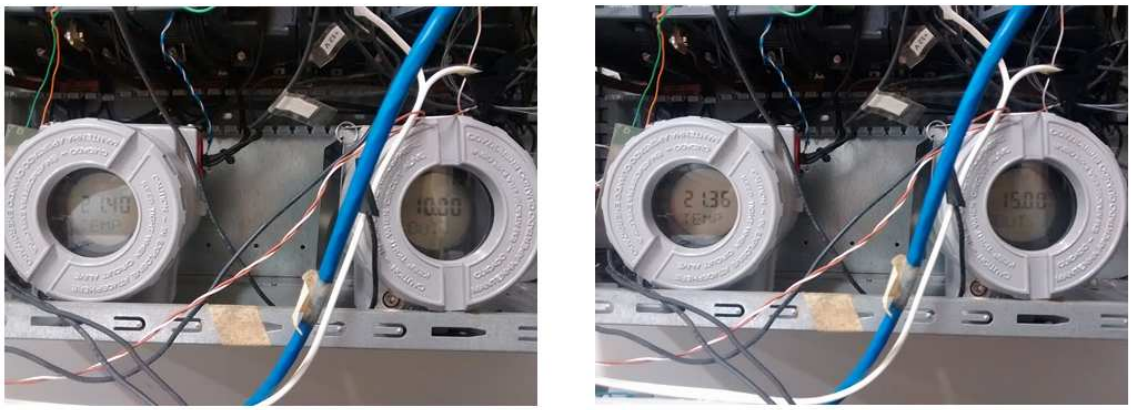
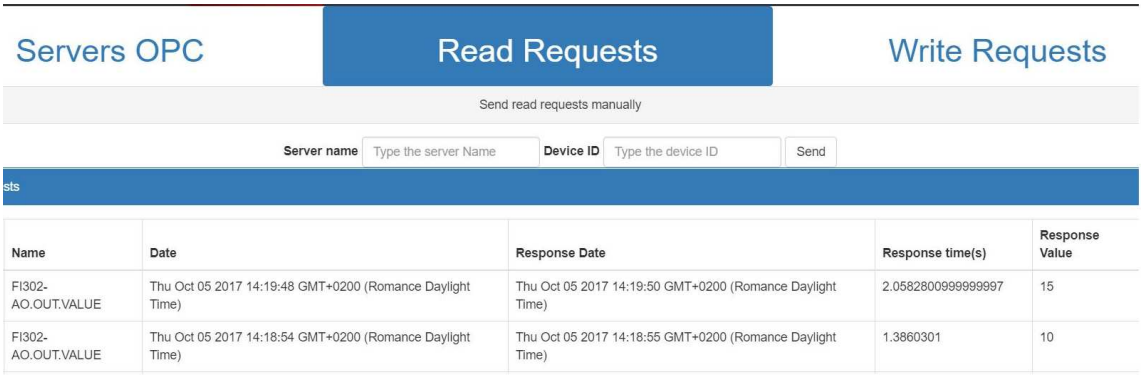


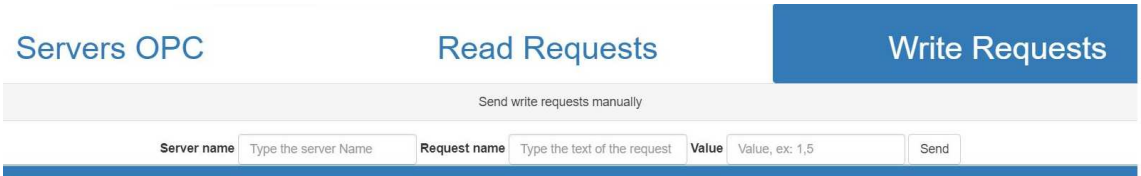
Figura 41. Estados dos visores dos dispositivos após realização dos pedidos de escrita e leitura,

Nas abas em que se encontram as 10 útimas requisições de leitura e escrita, dispostas, respectivamente, na **Figura 42** e **Figura 43** podem ser vistos com mais detalhes os dados dos pedidos analisados. O tempo para resposta destas requisições esteve entre 1 e um pouco mais de 2 segundos.



Name	Date	Response Date	Response time(s)	Response Value
FI302-AO.OUT.VALUE	Thu Oct 05 2017 14:19:48 GMT+0200 (Romance Daylight Time)	Thu Oct 05 2017 14:19:50 GMT+0200 (Romance Daylight Time)	2.0582800999999997	15
FI302-AO.OUT.VALUE	Thu Oct 05 2017 14:18:54 GMT+0200 (Romance Daylight Time)	Thu Oct 05 2017 14:18:55 GMT+0200 (Romance Daylight Time)	1.3860301	10

Figura 42. Detalhes dos dois pedidos de leitura feitos.



Name	Date	Response Date	Response time(s)	Value Sent	Response
FI302-AO.OUT.VALUE	Thu Oct 05 2017 14:19:46 GMT+0200 (Romance Daylight Time)	Thu Oct 05 2017 14:19:47 GMT+0200 (Romance Daylight Time)	1.6994051	15	Write succeed
FI302-AO.OUT.VALUE	Thu Oct 05 2017 14:18:47 GMT+0200 (Romance Daylight Time)	Thu Oct 05 2017 14:18:48 GMT+0200 (Romance Daylight Time)	1.5770301	10	Write succeed

Figura 43. Detalhes dos dois pedidos de escrita feitos.

3.3 Exemplo: servidor OPC da Schneider Electric

Neste teste, foi utilizado um servidor OPC da *Schneider Electric*, com licença de uso pertencente a ENSEM, e que é capaz de se comunicar com os CLP's da família TSX. O servidor já estava configurado para se ter acesso às sete variáveis de um módulo TSX Micro que fazem um controle PID de velocidade em um motor de corrente contínua. A planta citada pode ser vista na **Figura 44** e está em um dos laboratórios do Centro de Pesquisa em Automação de Nancy, localizado na ENSEM **[27].**

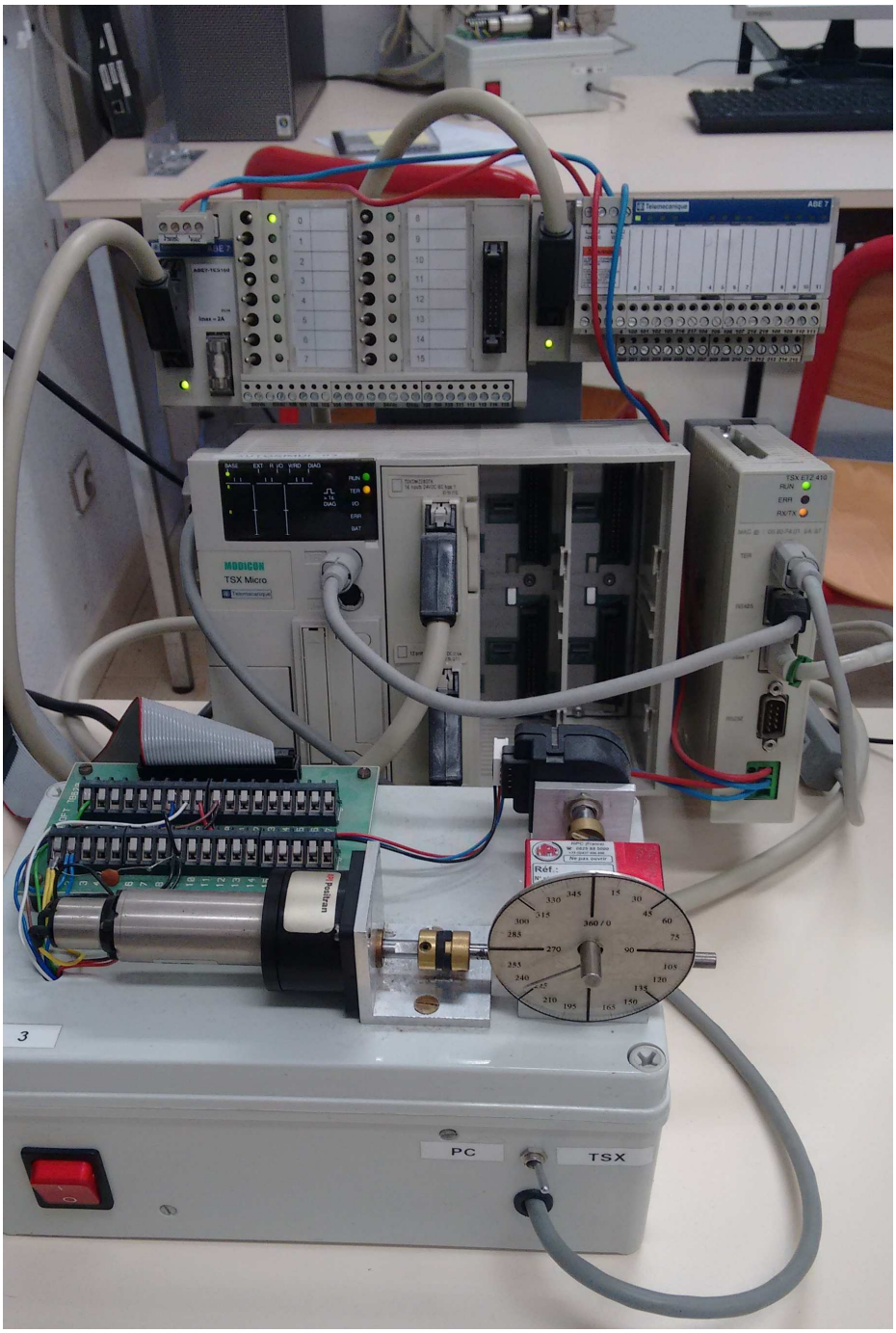


Figura 44. Planta utilizada para o teste desta seção.

Nas variáveis do CLP que foram identificadas, é possível obter as seguintes informações sobre a planta: valor da velocidade, valor da referência de velocidade, sinal de controle, ganho proporcional, constante de tempo da ação integral, constante de tempo da ação derivativa e o tempo de amostragem do controlador.

O servidor em questão foi executado em uma máquina virtual com sistema operacional *Windows 7 – 64 bits* como *guest* e o mesmo sistema operacional como *host*, sendo oferecido acesso à internet ao *guest.* A utilização da máquina virtual neste caso se deve a políticas do laboratório. Em resumo, para que se possa trabalhar com direitos de administrador no *guest* sem que seja necessário fornecê-los ao *host*, os quais são geralmente restritos para os alunos.

Como pode ser verificado na **Figura 45**, o *Cliente OPC* foi executado, a conexão com o servidor realizada e, de maneira similar aos exemplos anteriores, os identificadores das sete variáveis menciodadas anteriormente foram registrados no banco de dados.

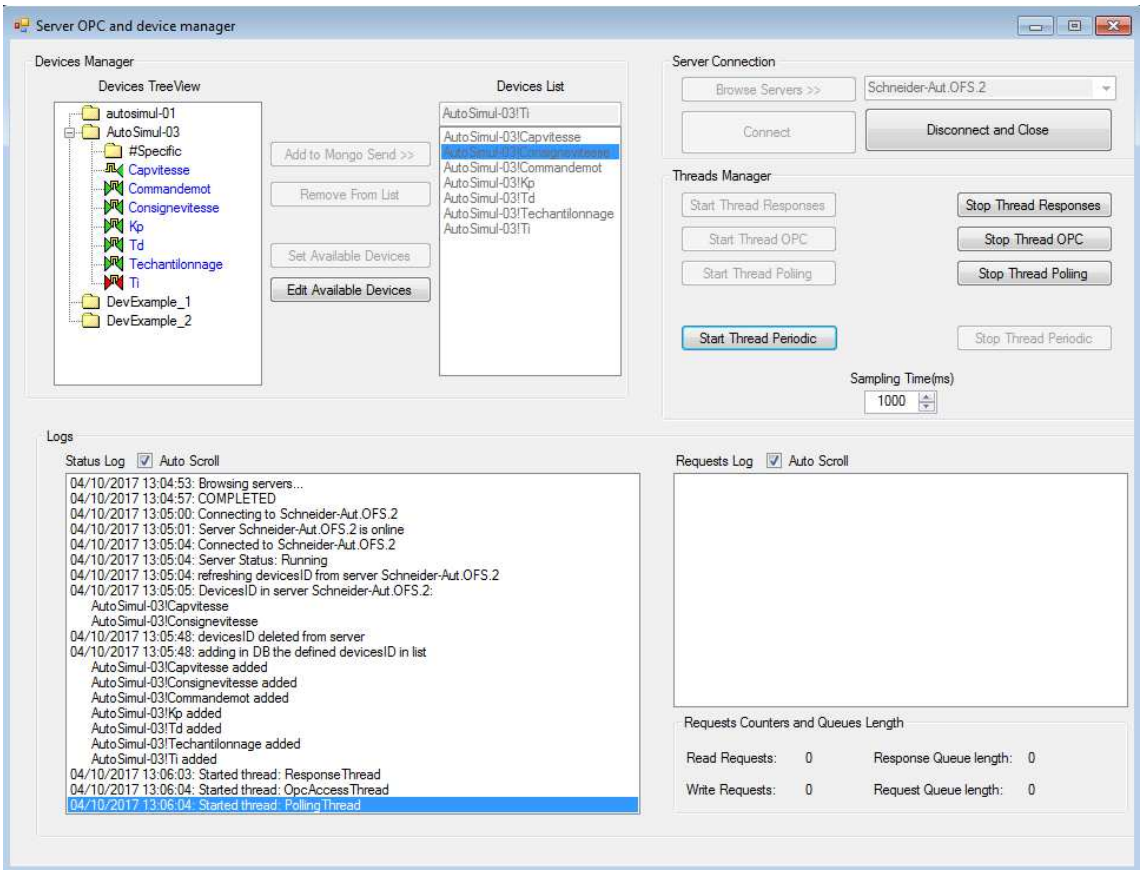


Figura 45. Estado da interface após conexão com o servidor e registro dos sete dispositivos.

Utilizando um celular com acesso à internet via 3G, o site foi acessado pelo navegador *Google Chrome*. Este acesso, bem como a disponibilidade dos sete dispositivos registrados, pode ser conferido na **Figura 46**.

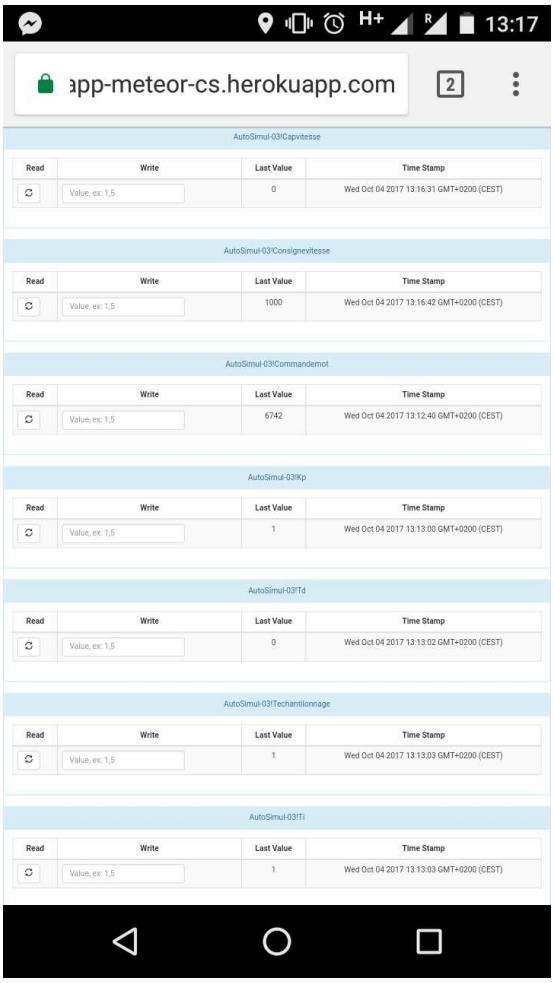


Figura 46. Acesso à aplicação pelo celular e identificação do registro dos sete dispositivos.

O motor foi ligado e, pelo celular, foi executado o pedido de leitura periódica clicando-se no botão *"Start"*. A partir deste momento, dados dos sete dispositivos começaram a ser registrados no banco de dados com o período de um segundo, definido no *Cliente OPC*. A atualização destes valores pôde ser monitorada pelo celular.

Após o início da aquisição de dados, o motor já se encontrava ligado e estabilizado na velocidade de valor representada por 1000. A escala desta variável do CLP começa em 0 e termina em 10000, sendo que os extremos representam as velocidades máximas que podem ser atingidas para cada sentido de rotação do motor. O valor intermediário desta escala, ou seja, 5000, representa o motor parado.

A seguir, foram feitos comandos de escrita na variável de referência e na variável do ganho proporcional. A condição incial do sistema era: valor da velocidade próximo a 1000,

valor de de referência igual a 1000 e o valor do ganho proporcional igual a 1. A sequência de ações foi:

- Realizou-se um pedido de escrita na variável de referência com valor de 2000, desacelerando o motor;
- Dada a atualização automática dos dados devido à leitura periódica em execução, esperou-se até que o valor da velocidade lida atingisse próximo o da referência de 2000;
- Pediu-se uma nova escrita na referência de valor 1000, acelerando o motor;
- Foi aguardado até que o sistema de controle atuasse, retornando o motor à sua velocidade inicial;
- Realizou-se um pedido de escrita no ganho proporcional com o valor de 5;
- Os quatro primeiros itens foram repetidos;
- Realizou-se um pedido de escrita no ganho proporcional com o valor de 20;
- Novamente, executou-se as ações contidas nos quatro primeiros itens.

A sequência de requisições de escrita realizadas com seus respectivos horários pode ser verificada no *Status Log* gerado pelo *Cliente OPC*, disposto na **Figura 47**. Também pode ser visto o momento em que é identificado o pedido de início da leitura periódica.

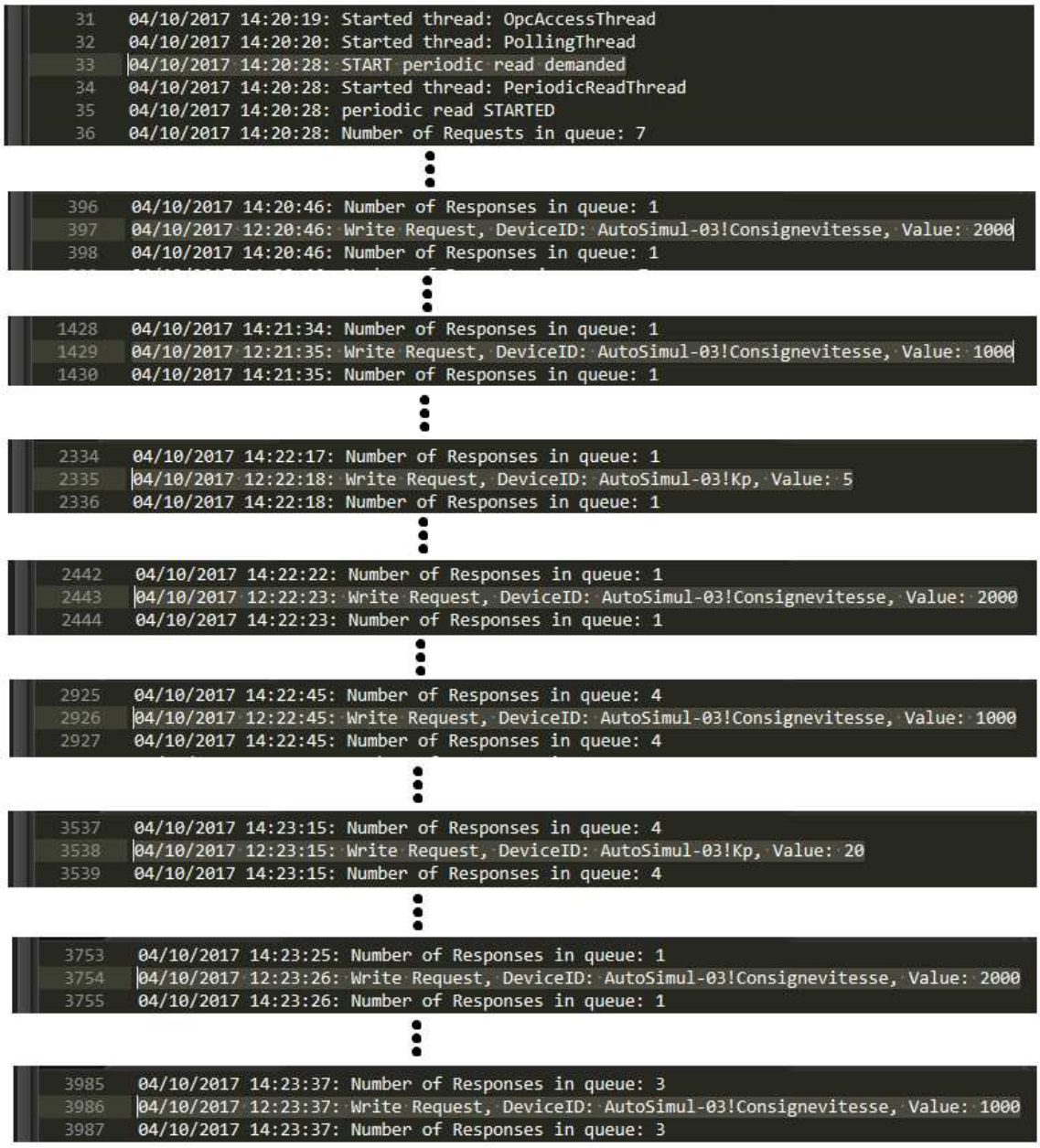


Figura 47. Arquivo .txt gerado: registro das requisições de escrita realizadas.

Com o arquivo *.csv* gerado, o qual possui os dados dos pedidos de leitura, e utilizando o *software Excel 2016,* foi contruído o gráfico da **Figura 48**.

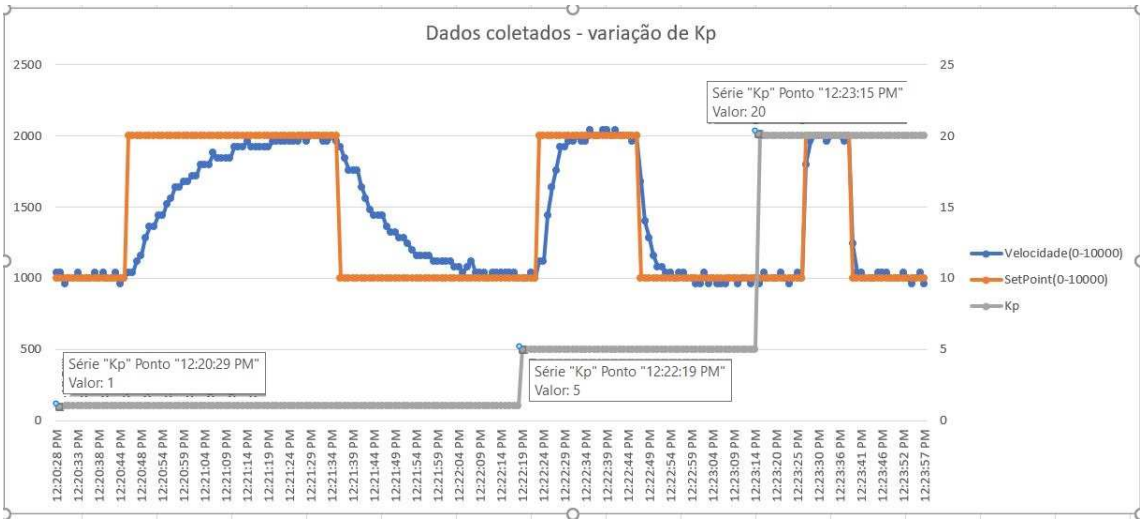


Figura 48. Gráfico gerado com os dados do arquivo .csv: valor da leitura x data da leitura

Cada ponto do gráfico representa uma requisição de leitura feita. O eixo horizontal representa o horário de tratamento da requisição. Já os eixos verticais representam os valores destas, sendo o da esquerda para a velocidade e referência e o da direita para o ganho proporcional.

Em destaque indicou-se o horário em UTC das leituras dos valores do ganho proporcional logo após a mudança de seu valor pelo pedido de escrita. Pode-se verificar, junto à **Figura 48**, que as datas são muito próximas, diferenciando-se em no máximo 1 segundo.

São verificados os comportamentos já esperados para o sistema de controle em questão: a variável de referência é seguida pela variável medida e há uma maior velocidade de convergência ao passo que o ganho do controlador proporcional é maior. Verifica-se, portanto, uma consistência entre os dados coletados deste sistema.

4 Conclusões e trabalhos futuros

Um sistema capaz de acessar e escrever dados em dispositivos de plantas industriais por meio da internet foi feito neste trabalho. Foram empregadas tecnologias emergentes na área de desenvolvimento *web* como o *framework Meteor*, além de padrões presentes nas diversas indústrias como o padrão OPC.

O sistema fornece uma ferramenta para a gestão de dispositivos em uma rede industrial e registrados em um servidor OPC. Além do monitoramento e controle em tempo real, podem ser feitas análises do histórico das informações adquiridas, dada a geração de arquivos que salvam os detalhes da conexão com o servidor OPC.

Ao se utilizar um padrão aberto e extremamente difundido como o OPC e uma estrutura como a internet, foi possível ter versatilidade nas aplicações. Desta forma, diversas são as possibilidades de se empregar o sistema, desde experimentos didáticos que visam estudar sistemas dinâmicos até processos industriais.

Como propostas para trabalhos futuros podem ser citadas o desenvolvimento de segurança da informação adquirida, pois a mesma se encontra pública, e interfaces na *web application* que permitam ao usuário consultar em mais detalhes as leituras gravadas no banco de dados. Para o cliente OPC, é importante desenvolver um tratamento para eventuais problemas de comunicação relacionados ao acesso ao banco de dados na internet.

Referências Bibliográficas

[1] STAIR, Ralph M.; REYNOLDS, George W.. Introdução aos Sistemas de Informação. In: STAIR, Ralph M.; REYNOLDS, George W.. **Princípios de sistemas de informação**: uma abordagem gerencial. 4. ed. Rio de Janeiro: Ltc, 2004. p. 2.

[2] LAFORE, Robert. Object-Oriented Programming. In: LAFORE, Robert. **Object-Oriented Programming in C++**: Fourth Edition. 4. ed. Indiana: Sams, 2002. p. 1.

[3] BORGES, Roberto; CLINIO, André Luiz. **Programação Orientada a Objetos com C++**.2017. Disponível em: <http://webserver2.tecgraf.puc-rio.br/ftp_pub/lfm/CppBorgesClinio.pdf>. Acesso em: 11 nov. 2017. p. 12-13

[4] BUYYA, Rajkumar. DEFINING THREADS. In: BUYYA, Rajkumar; SELVI, S Thamarai; CHU, Xingchen**. Object-Oriented Programming with Java**: Essentials and Applications. Noida: Mc Graw Hill, 2009. Cap. 14, p. 365.

[5] ARPACI-DUSSEAU, Remzi H.. Concurrency: An Introduction. In: ARPACI-DUSSEAU, Remzi H.; ARPACI-DUSSEAU, Andrea C. **Operating Systems**: Three Easy Pieces. Madison: Arpaci-dusseau Books, Llc, 2016. p. 2-3.

[6] MCCREESH, John; DANIEL, Ed. **Definition of Interoperability**. Disponível em: <http://interoperability-definition.info/en/>. Acesso em: 11 nov. 2017.

[7] OPC FOUNDATION; **The Interoperability Standard for Industrial Automation**. Fundação de divulgação das características do padrão OPC. Disponível em: <opcfoundation.org>. Acesso em: 11 nov. 2017.

[8] RUST, Sophie; SCHELLING, Jasper; SCHIPPER, Donna. **Building Real-Time Web Applications with Meteor**. Disponível em: <http://mediatechnology.leiden.edu/images/uploads/docs/wt2015_meteor.pdf>. Acesso em: 11 nov. 2017.

[9] METEOR; **Meteor web development framework**, 2012. Disponível em: <https://www.meteor.com/>. Acesso em: 11 nov. 2017.

[10] VERACODE; **WHAT IS AN INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)?** Disponível em: <https://www.veracode.com/security/integrated-development-environments>. Acesso em: 11 nov. 2017.

[11] MICROSOFT; **Guia de Introdução ao Visual Studio**. Disponível em: < https://msdn.microsoft.com/pt-br/library/ms165079(v=vs.110).aspx>. Acesso em: 11 nov. 2017.

[12] ADVOSOL INC.; **Advosol specializes in OPC solutions on the .NET platform**. Disponível em: < http://www.advosol.com/topic/about>. Acesso em: 11 nov. 2017.

[13] ADVOSOL INC.; **OPC DA .NET Client Development Toolkit for C# and VB.NET**. Disponível em: <http://www.advosol.com/product/1/opcda-net#ProductInfo>. Acesso em: 11 nov. 2017.

[14] MONGODB UNIVERSITY; **The next generation .NET driver for MongoDB**. Disponível em: <https://mongodb.github.io/mongo-csharp-driver/>. Acesso em: 11 nov. 2017.

[15] MONGODB UNIVERSITY; **MongoDB Drivers.** Disponível em: < https://docs.mongodb.com/ecosystem/drivers/>. Acesso em: 11 nov. 2017.

[16] BANKER, Kyle. 2011. **MongoDB in Action**. Manning Publications Co., Greenwich, CT, USA.

[17] ECMA INTERNATIONAL. 2013. **The JSON Data Interchange Format.** Disponível em: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>. Acesso em: 11 nov. 2017.

[18] MLAB; **mLab:** Database-as-a-Service for MongoDB. Disponível em: < https://mlab.com/company/>. Acesso em: 11 nov. 2017.

[19] SADJADEE, Sahand**. Meteor framework, a new approach to web development**: an experimental analysis. 2013. 41 f. Monografia (Especialização) - Curso de Information Science, Institutionen För Datavetenskap, Linköping, 2013.

[20] BOOTSTRAP; **The most popular HTML, CSS, and JavaScript framework for developing responsive, mobile first projects on the web**. Disponível em: < http://getbootstrap.com/>. Acesso em: 11 nov. 2017.

[21] PERCOLATE STUDIO; **The trusted source for JavaScript packages, Meteor resources and tools | Atmosphere**. Disponível em: < https://atmospherejs.com/>. Acesso em: 14 nov. 2017.

[22] HEROKU ENTERPRISE; **Cloud Application Platform | Heroku**. Disponível em: <https://www.heroku.com/>. Acesso em: 14 nov. 2017.

[23] SMAR; **System302:** GUIA DE INSTALAÇÃO, 2009.

[24] SMAR; **Fieldbus Universal Bridge:** MANUAL DO USUÁRIO, 2008.

[25] SMAR; **Conversor Fieldbus para Corrente com Três Canais:** MANUL DE INSTRUÇÕES, OPERAÇÕES E MANUTENÇÃO, 2007.

[26] SMAR; **TRANSMISSOR DE TEMPERATURA FIELDBUS:** MANUAL DE INSTRUÇÕES, OPERAÇÃO E MANUTENÇÃO, 2006.

[27] SCHNEIDER ELECTRIC; **OPC Factory Server V3.0:** Manuel utilisateur, VTLX DM OFS 3.0.