**UNIVERSIDADE DE CAXIAS DO SUL – UCS**
**CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA – CCET**
**CIDADE UNIVERSITÁRIA**
**ENGENHARIA DE CONTROLE E AUTOMAÇÃO**

**GUILHERME DOS SANTOS THOMÉ**

# PROJETO E APLICAÇÃO DE UM SISTEMA DE COLETA DE DADOS EM TEMPO REAL PARA INDÚSTRIA

**CAXIAS DO SUL**
**2015**

**GUILHERME DOS SANTOS THOMÉ**

# PROJETO E APLICAÇÃO DE UM SISTEMA DE COLETA DE DADOS EM TEMPO REAL PARA INDÚSTRIA

Trabalho de conclusão de curso de graduação, apresentado ao Centro de Ciências Exatas da Natureza e de Tecnologia da Universidade de Caxias do Sul, como requisito para a obtenção do grau de Engenheiro de Controle e Automação.

Orientador: Prof. Dr. André Gustavo Adami

**CAXIAS DO SUL**

**2015**

**GUILHERME DOS SANTOS THOMÉ**

# PROJETO E APLICAÇÃO DE UM SISTEMA DE COLETA DE DADOS EM TEMPO REAL PARA INDÚSTRIA

Trabalho de conclusão de curso de graduação, apresentado ao Centro de Ciências Exatas da Natureza e de Tecnologia da Universidade de Caxias do Sul, como requisito para a obtenção do grau de Engenheiro de Controle e Automação.

Aprovado em _____ de _____ de _____.

**Banca Examinadora**

_____

Professor Dr. André Gustavo Adami
Universidade de Caxias do Sul – UCS

_____

Professor Dr. Mauricio Zardo Oliveira
Universidade de Caxias do Sul – UCS

_____

Professor Dr. Renato Gonçalves Ferraz
Universidade de Caxias do Sul - UCS

## AGRADECIMENTOS

A Deus.

À minha família, pelo apoio e incentivo.

Ao meu orientador Prof. Dr. André Gustavo Adami pelas ideias, sugestões e incentivos que foram indispensáveis neste trabalho.

Aos meus amigos e colegas de curso pelos momentos de apreensão e diversão ao longo do curso.

## RESUMO

A globalização tem impulsionado as indústrias a melhorar sua eficiência e buscar inovação. O controle das informações de produção em tempo real permite aos gestores avaliar e tomar decisões sobre o processo produtivo que refletem na eficiência da empresa. Com foco na comunicação em tempo real das informações da empresa, este trabalho apresenta um sistema de comunicação sem fio para máquinas industriais a fim de disponibilizar informações produtivas em tempo real. O sistema é composto por um dispositivo para coletar dados de uma máquina, no qual opera em conjunto com um *tablet* responsável pela *interface* com o operador. Ambos são conectados ao servidor da empresa através de um roteador Wi-Fi. O servidor possui programas para comunicação com os dispositivos da fábrica, armazenamento em um banco de dados, e provimento de consultas das informações pelos gestores da indústria. Estas consultas são requisitadas via HTTP por navegadores de internet dos *desktops* ou dispositivos portáteis dos gestores. Um sistema protótipo foi desenvolvido para validação técnica e testes dos recursos propostos.



**Palavras-chave:** IIoT, rede Wi-Fi, coletor de dados, dispositivos móveis, servidor web.

## ABSTRACT

Globalization has inspired the industry to improve its efficiency and to search for innovation. Real-time information control in Manufacturing allows managers to evaluate and make decisions about the manufacturing process that reflect the company's efficiency. Focused on real-time communication of company information, this paper presents a wireless communication system for industrial machinery in order to provide manufacturing information in real time. The system consists of a device to collect data from a machine, which operates in conjunction with a tablet responsible for interacting with the operator. Both are connected to the company server through a Wi-Fi router. The server has software for communication with the factory devices, managing a database, and retrieving information through queries by industry managers. These queries are requested via HTTP for Web browsers of desktops or handheld devices of managers. A prototype system was developed for technical validation and testing of the proposed resources.



**Keywords:** IIoT, Wi-Fi network, data collector, mobile devices, web server.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE CÓDIGOS

## LISTA DE SIGLAS

| | |
|---|---|
| API | *Application Programming Interface* |
| BD | Banco de Dados |
| APK | *Android Package* |
| ARM | *Advanced RISC Machine* |
| DHCP | *Dynamic Host Configuration Protocol* |
| GPIO | *General Purpose Input/Output* |
| HAL | *Hardware Abstraction Layer* |
| IDE | *Integrated Development Environment* |
| IHM | Interface Homem-Máquina |
| JDBC | *Java Database Conectivity* |
| JDK | *Java Development Kit* |
| M2H | *Machine to Human* |
| M2M | *Machine to Machine* |
| MCU | *Microcontroller Unit* |
| OP | Ordem de Produção |
| RISC | *Reduced Instruction Set Computer* |
| SDK | *Software Development Kit* |
| SO | Sistema Operacional |
| SQL | Linguagem de Consulta Estruturada |
| UART | *Universal Asynchronous Receiver and Transmitter* |

**SUMÁRIO**

# 1 INTRODUÇÃO

Devido à competitividade dentro do cenário globalizado, as indústrias buscam evoluir seus processos a fim de alcançar melhores resultados como: aumento de produtividade, economia de energia, maior qualidade do produto e redução de desperdícios. Para isso é necessário um aumento no número de informações produtivas e controle dos processos (CASSIOLATO, 2011).

A principal necessidade dos empresários é a tomada de decisões baseadas em informação em tempo real. Aplicações industriais como monitoramento remoto das máquinas, controle de processos, rastreamento de peças, entre outras, têm como principal característica a comunicação de dados em tempo real aliado à confiabilidade das informações. Caso estes requisitos não forem atendidos, poderá resultar em perda de tempo, qualidade e dinheiro para a empresa (WILLIG, 2008).

Em conjunto com a automação industrial, as tecnologias de comunicação sem fio (*wireless*) estão sendo identificadas como opções muito atraentes para a indústria. Pois elas proporcionam benefícios como: flexibilidade, escalabilidade, facilidade e rapidez na instalação, e movimentação na atual indústria, na qual ocorrem mudanças frequentes no laiout fabril. A tecnologia *wireless* começa a ser observada com mais atenção na indústria em função dos benefícios que pode fornecer (GINATTO e RAPHALOSKI, 2011).

Através da necessidade de automação industrial sem fio surgiu a *Industrial Internet of Things* (IIoT), que é o uso da tecnologia da Internet das Coisas (IoT) na indústria. A IIoT conecta dispositivos (como equipamentos e sensores industriais) na rede, permitindo coletar dados e gerenciar as informações destes dispositivos via *software* para aumentar a eficiência e produtividade da fábrica. Neste sistema, as máquinas, sensores e outras tecnologias operacionais se comunicam entre si, formando uma arquitetura inteligente dentro das organizações para processamento de informações e tomada de ações em tempo real (Accenture, 2015).

Este trabalho apresenta um estudo, projeto e implementação de um sistema de comunicação sem fio em tempo real para máquinas industriais. Um dispositivo é desenvolvido para coletar dados de uma máquina industrial. Um *tablet* é utilizado pelo operador da máquina para visualização das informações de produção e

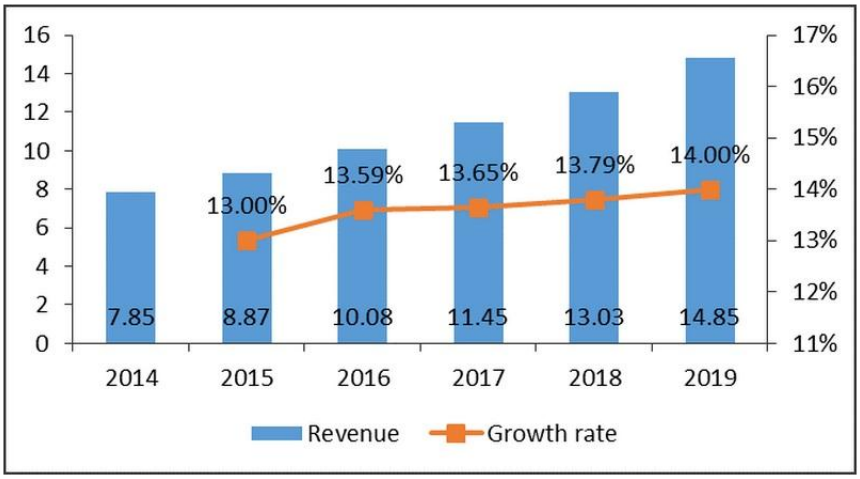
interação com a máquina. Ambos são conectados via Wi-Fi na mesma rede de um servidor, que deve armazenar e processar os dados do dispositivo coletor e *tablet*. Os gestores utilizam *desktops* ou dispositivos portáteis para gerenciamento em tempo real das informações.

## 1.1 Justificativa do Trabalho

Pesquisas publicadas pela TechNavio – empresa líder de pesquisa e consultoria de tecnologia – apontam um crescimento no mercado global de Internet da Coisas de 31,72% entre 2014 e 2019. O novo relatório chama atenção para o número crescente de dispositivos inteligentes conectados, que passará de 17 bilhões nos últimos cinco anos. As comunicações M2M (*Machine-to-Machine*) – que permite a interação segura das máquinas em uma rede – e M2H (*Machine-to-Human*) – que permite a interação da máquina com o homem através de uma interface – interconectadas a tecnologia IoT irão aumentar a demanda nos próximos anos (TechNavio, 2015).

Analistas estimam que o mercado global de automação industrial tenha uma taxa de crescimento anual de 11% entre 2015 e 2019, devido ao aumento da procura de soluções em todos os setores. A Figura 1.1 apresenta um gráfico com o crescimento global do mercado de automação industrial durante os anos de 2014 a 2019. As barras azuis apresentam a receita gerada em bilhões de dólares, enquanto a linha laranja demonstra a taxa de crescimento (TechNavio, 2015).

Figura 1.1: Mercado da automação industrial.




*Fonte: TechNavio (2014).*

Enquanto cresce o investimento em automação industrial, as pessoas também estão cada vez mais familiarizadas a dispositivos móveis e à Internet. As redes sem fio com tecnologia Wi-Fi se tornaram uma realidade presente nos ambientes domiciliares e corporativos. Facilidade de instalação e acesso, baixo custo e juntamente com a popularização de dispositivos móveis, impulsionam esta tecnologia (RIBEIRO, AMADIO, *et al.*, 2012). Segundo o estudo trimestral da IDC (2015) sobre o mercado de roteadores no Brasil, no primeiro trimestre de 2014 a receita com esses equipamentos foi de US$ 63 milhões, o que representa um crescimento de 10,2% em relação ao primeiro trimestre de 2013.

Atrelando a necessidade do mercado pelo aumento no número de informações produtivas e controle dos processos, com a larga utilização de *tablets* e redes Wi-Fi, o mercado da automação se torna viável e incentivador. O desenvolvimento proposto neste trabalho busca estudar as tecnologias expostas, como a comunicação Wi-Fi, o desenvolvimento de aplicativos para dispositivos Android, servidores e banco de dados, entre outras. Contudo, possui a finalidade de projetar e implementar um sistema baseado na arquitetura IIoT para disponibilização de informações produtivas na indústria.

## 1.2 Objetivos

Esta seção descreve o objetivo geral e os objetivos específicos para o desenvolvimento do presente trabalho.

### 1.2.1 Objetivo Geral

Estudo, projeto e implementação de um sistema para coletar e gerenciar dados de uma máquina industrial em tempo real.

### 1.2.2 Objetivos Específicos

- Especificar o hardware do dispositivo coletor;
- Desenvolver o *firmware* do dispositivo coletor para o microcontrolador ARM;
- Desenvolver um aplicativo para *tablets* com sistema operacional Android utilizados pelos operadores das máquinas;

- Desenvolver o *software* do servidor para o armazenamento e processamento dos dados;
- Definir um protocolo de comunicação para troca de mensagens via Socket;
- Desenvolver um servidor web para permitir consultas das informações de produção aos dispositivos dos gestores.
- Preparar um sistema protótipo para testes funcionais e validação técnica;

## 1.3 Escopo e Restrições

São aplicadas algumas restrições no desenvolvimento deste trabalho a fim de atingir todos os objetivos propostos:

a) Para testes do hardware do sistema serão utilizados apenas equipamentos comerciais e kits de desenvolvimento, ou seja, não será desenvolvido nenhum dispositivo específico.
b) Os testes devem garantir a funcionalidade do sistema, não prevendo características como infraestrutura da rede Wi-Fi, perdas de pacote, entre outros.

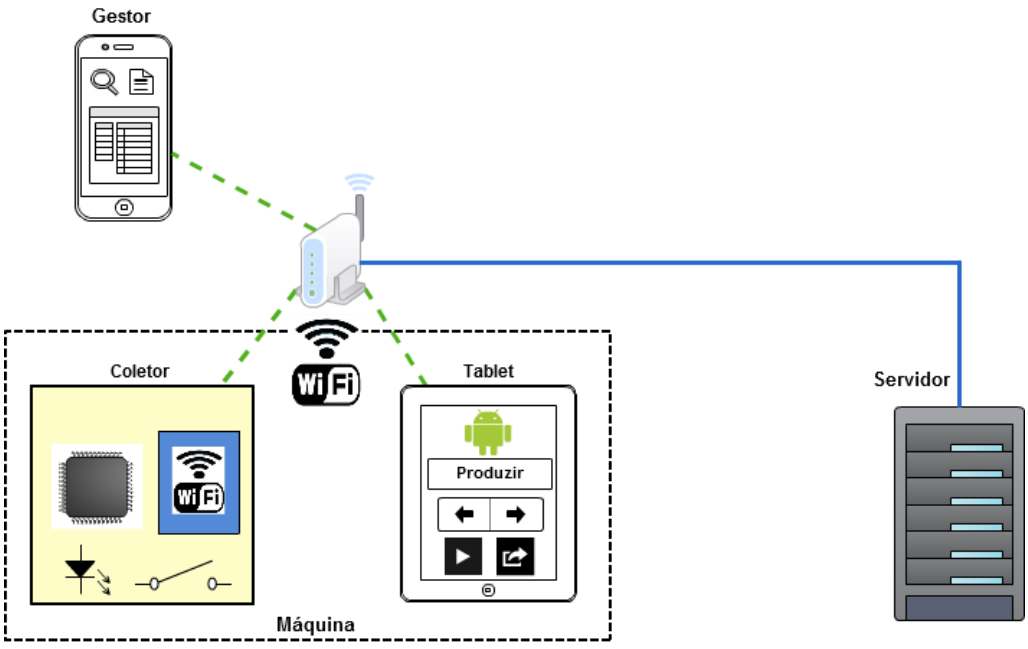
# 2 SISTEMA DE COLETA DE DADOS PARA MÁQUINAS INDUSTRIAIS

Este capítulo descreve os procedimentos realizados para execução dos objetivos propostos neste trabalho. Na Seção 2.1, a arquitetura do sistema é apresentada, assim situando sobre o conceito geral para depois se aprofundar nos itens individuais. A Seção 2.2 apresenta o microcontrolador utilizado e a interface de desenvolvimento do mesmo. A Seção 2.3 descreve a programação do *firmware* do coletor. A Seção 2.4

## 2.1 Arquitetura do Sistema

O sistema é composto por três partes, conforme a Figura 2.1: o *tablet* operando em conjunto com o coletor para interagir com a máquina, o servidor para armazenar e processar os dados coletados e, por fim, um dispositivo móvel para visualizar as informações produtivas. Todos dispositivos móveis são conectados à mesma rede sem fio Wi-Fi através de um roteador. Os *tablets* e *smartphones* já possuem suporte para tecnologia Wi-Fi, enquanto no coletor foi implementado através de um módulo Wi-Fi. O servidor é conectado ao roteador através de um cabo de rede e tem como função centralizar todas as comunicações existentes (todos outros dispositivos se comunicam diretamente com o servidor) e fazer o processamento dos dados.

Figura 2.1: Arquitetura da rede.




*Fonte: O AUTOR (2015).*

O coletor é responsável por obter informações da máquina, como por exemplo, o estado da máquina (em operação ou parada), e ainda poderá interferir no funcionamento dela, bloqueando-a quando necessário. O coletor possui entradas e saídas para a integração com a máquina industrial. O *tablet* é a IHM (Interface Homem Máquina) utilizada pelo operador da máquina, que deve funcionar em conjunto com o coletor para que o operador possa visualizar informações de produção e interagir com a máquina.

As funções do servidor são armazenar os dados fornecidos pelos coletores e *tablets* de diferentes máquinas e posteriormente quando solicitado pelos dispositivos móveis dos gestores, processar os dados e enviar as informações requeridas. O servidor possui uma interface de comunicação via Sockets TCP/IP para enviar e receber os dados dos dispositivos móveis.

Para o gerenciamento das informações já processadas pelo servidor, um computador ou dispositivo móvel solicita ao servidor, que por sua vez enviará o conteúdo solicitado em HTML (*Hyper Text Language Markup*). Como a interface com o servidor será através do HTML, toda informação poderá ser visualizada tanto em computadores, quando em *smartphones* e *tablets* de diferentes fabricantes.

## 2.2 Microcontrolador

Um microcontrolador (MCU) é responsável por controlar o coletor de dados, reunindo informações e permitindo o controle remoto da máquina (neste trabalho será apenas através de entradas e saídas, e futuramente poderá ser implementado outras interfaces de comunicação). Em seguida o MCU deve comunicar via serial-UART (*Universal Asynchronous Receiver and Transmitter*) com o módulo Wi-Fi, enviando/recebendo comandos para/do servidor.

Para escolha do MCU devem ser levados em consideração os periféricos requeridos e velocidade de processamento, associando custo e autonomia. Dentre os núcleos de microcontroladores atuais no mercado, o mais utilizado em sistemas embarcados é o ARM (*Advanced RISC Machine*), empregado por grande parte dos fabricantes de microcontroladores. A arquitetura ARM utiliza um conjunto de instruções para processadores de 32 bits baseados na arquitetura RISC (*Reduced Instruction Set Computer*), os quais são projetados para alcançar um bom equilíbrio

de desempenho, simplificação de instruções, consumo energéticos eficientes e custos competitivos (ARM, 2015).

Optou-se por utilizar um microcontrolador da família Cortex-M0, que é o menor processador ARM disponível. Os benefícios de baixo consumo energético e densidade de código do Cortex-M0 indicam que ele é um sucessor natural para os MCUs de 8 e 16 bits em uma ampla variedade de aplicações sensíveis ao custo. E ainda mantém ferramentas e compatibilidade binária com processadores mais poderosos como o Cortex-M3 e Cortex-M4 (ARM, 2015).

Para a seleção do microcontrolador, foram analisadas as soluções fornecidas pela fabricante de semicondutores STMicroeletronics (ST)[1], qual já foi utilizado em trabalhos anteriores de graduação (THOMÉ, 2015). Ela fornece diversas soluções para sistemas embarcados, contendo uma abrangente variedade de microcontroladores com núcleo ARM, a qual é chamada de família STM32. A Figura 2.2 apresenta a ampla família de microcontroladores deste fabricante. Foram avaliados os MCUs da ST que apresentam os recursos necessários para atingir os objetivos do projeto, tais como, conversor analógico-digital, comunicação serial UART, temporizadores e alguns pinos de entradas e saídas digitais.

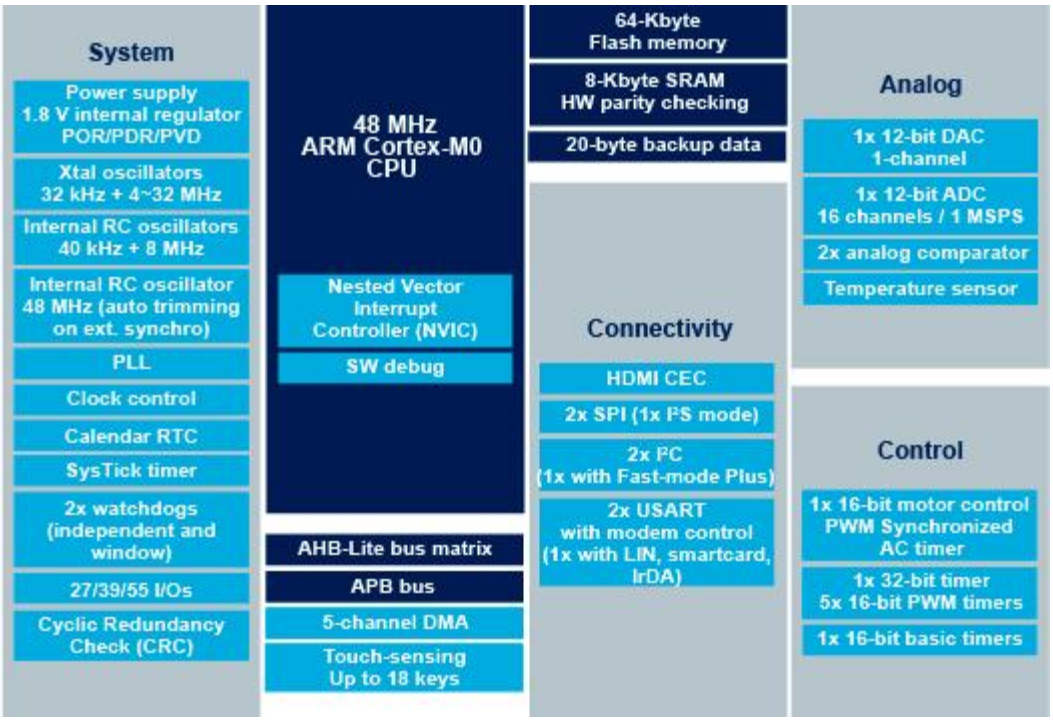
Figura 2.2: Família de microcontroladores STM32.


*Fonte: ST (2015).*

[1] http://www.st.com

O microcontrolador da série STM32F051 foi selecionado para este projeto. Ele opera a uma frequência de até 48 MHz, a memória Flash pode chegar até 64 Kbytes e 8 Kbytes de SRAM (*Static Random Access Memory*), e contém uma extensa gama de periféricos e GPIOs (*General Purpose Input/Output*). O mesmo apresenta diversos periféricos extras que não serão usados neste projeto, porém em próximas oportunidades podem ser adicionados sem substituição do controlador. Além de que a ST presa muito pela escalabilidade e compatibilidade entre seus microcontroladores, proporcionando maior flexibilidade aos projetos. A Figura 2.3 demonstra os principais recursos deste MCU.

Figura 2.3: Recursos do microcontroladores STM32F051.




*Fonte: ST (2015).*

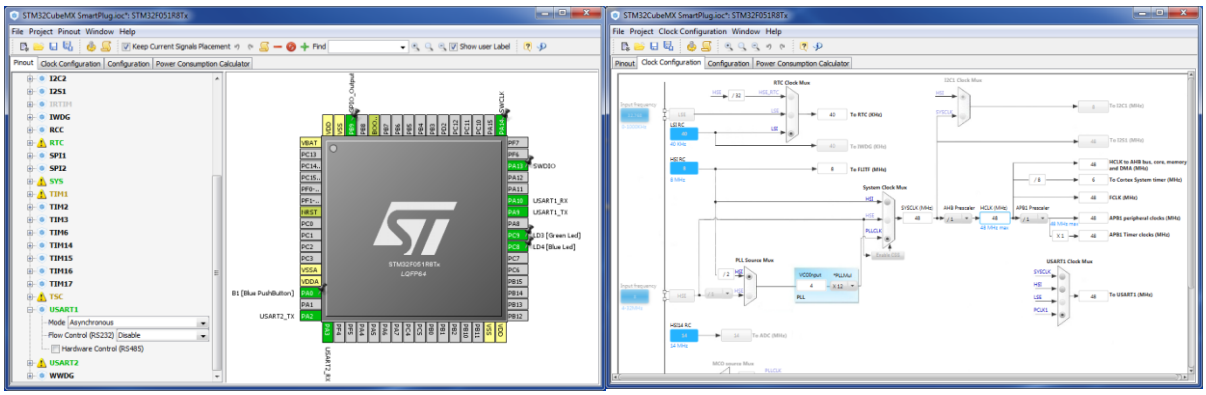
O kit de desenvolvimento STM32F0DISCOVERY, que a ST fornece para explorar os recursos deste microcontrolador, foi utilizado na implementação deste projeto. O mesmo utiliza o modelo de microcontrolador STM32F051R8T6, com encapsulamento LQFP e 64 pinos. Este kit ainda possui conector mini USB para gravação e depuração facilitada, LEDs e botões inclusos, além de compatibilidade com outros kits de desenvolvimento da ST.

## 2.3 *Firmware* do Coletor

O *firmware* da tomada eletrônica é feito em linguagem C, e foi estruturado em duas partes. A primeira é o código inicial, para configuração dos periféricos e clocks, e a segunda é a aplicação e lógicas do coletor.

O *software* STM32Cube da STMicroelectronics foi utilizado para gerar o código de inicialização do MCU. É uma ferramenta gráfica para pré-configuração do microcontrolador, onde é possível determinar os clocks, configurar os pinos utilizados, configurar os periféricos e até calcular o consumo do MCU (STM32Cube, 2015). A Figura 2.4 ilustra a interface de configuração do STM32Cube, na qual são configurados e habilitados os periféricos do microcontrolador, e definidas todas as frequências de operação da CPU e dos periféricos, respectivamente. Também há outras interfaces de configuração mais específicas para os periféricos individuais, uso de interrupções, e até consumo e tempo de duração quando uma bateria for utilizada na alimentação dos circuitos

Figura 2.4: Telas do *software* STM32Cube.



*Fonte: STM32Cube (2015).*

Os periféricos configurados através do *software* STM32Cube foram: GPIO para leitura de entradas digitais e acionamento de saídas digitais; TIMER1 para base de tempo do *firmware*; UART1 para comunicação Wi-Fi. Após definida a configuração, o STM32Cube gera um projeto específico para um IDE (ambiente integrado de desenvolvimento), no qual será feita a programação da aplicação em linguagem C.

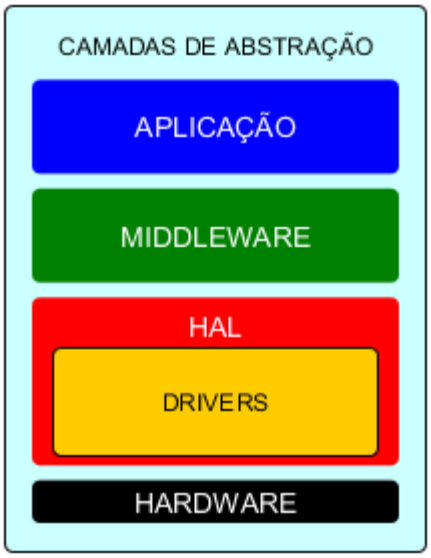
O projeto foi implementado utilizando a IDE MDK-ARM da companhia Keil. Este kit para desenvolvimento ARM (incluindo a família STM32) dispõe do ambiente para

programação e depuração µVision, do compilador ARM C/C++, além de outras ferramentas (KEIL).

### 2.3.1 Estrutura do *Firmware*

O *firmware* é organizado através da camada de abstração de hardware (HAL), middleware (fazem a mediação entre as camadas de hardware e aplicação) e aplicação do usuário. Na Figura 2.5 são visualizadas as camadas separadamente, onde na parte superior encontram-se as bibliotecas da aplicação e middleware que foram desenvolvidas, e na HAL as bibliotecas importadas automaticamente pelo *software* STM32Cube.

Figura 2.5: Camadas de abstração




*Fonte: O AUTOR (2015).*

A HAL é constituída pelos seguintes serviços:

- Drivers: proporcionam a manipulação do hardware;
- API (*Application Programming Interface*): fornece uma interface de acesso padronizado aos drivers. Possui uma documentação com descritivos dos recursos disponibilizados para a comunicação (ALTERA, 2015).

As principais bibliotecas de drivers utilizadas no projeto são UART, GPIO e outras de controle do clock, da memória e processador Cortex. Estes drivers têm o intuito de abstrair a camada de acesso pelo usuário da parte de hardware do projeto através das APIs. Caso necessário uma alteração do microcontrolador, basta

apenas adicionar os novos drivers, sem necessidade de alterar as lógicas da aplicação. Para exemplificação, é demonstrado no Código 2.1, como que foi configurado o uso de uma serial (UART2) conectada ao computador com a finalidade de depurar o *firmware*.

Código 2.1 – Inicialização UART2

```
UART_HandleTypeDef huart2;       // Declaração da estrutura de manipulação da UART2
huart2.Instance           = USART2;             // Definido UART2
huart2.Init.BaudRate      = 38400;              // Taxa de transmissão de dados
huart2.Init.WordLength    = UART_WORDLENGTH_8B; // Tamanho da palavra de dado
huart2.Init.StopBits      = UART_STOPBITS_1;    // 1 stop bit
huart2.Init.Parity        = UART_PARITY_NONE;   // Sem paridade
huart2.Init.Mode          = UART_MODE_TX_RX;    // Modo de transmissão full duplex
huart2.Init.HwFlowCtl     = UART_HWCONTROL_NONE;// Sem controle de fluxo via hardware
huart2.Init.OverSampling  = UART_OVERSAMPLING_8;// 8 amostras

HAL_UART_Init(&huart2);   // Inicialização da UART2
```

*Fonte: O AUTOR (2015).*

Todos os drivers seguem a mesma metodologia que é representada no Código 2.1, em que primeiramente declara-se uma estrutura de manipulação da UART no qual são contidos todos os registradores e as configurações necessárias para o funcionamento do driver. Após configurar os parâmetros de inicialização é chamada a API passando o ponteiro[2] da estrutura a ser criada. Percebe-se que o usuário não precisa ter muito contato com a programação em nível de hardware, pois as bibliotecas de HAL funcionam perfeitamente, bastando apenas entender seus parâmetros e quais as APIs necessárias para sua utilização, dentre as diversas oferecidas.

Para acesso a determinados periféricos, como a UART, são oferecidos três métodos: DMA (*Direct Memory Access*), interrupção, e programada. No método para acesso programada, o processador é responsável em ler repetidamente os registros de estado dos periféricos para determinar quando que estes se encontram prontos para iniciar uma transferência de dados. Assim, o processador fica travado (espera ocupada) executando a tarefa até que o periférico esteja pronto (STALLINGS, 2010).

Na interrupção, segundo STALLINGS (2010), o controlador envia ao processador um sinal indicando alterações no seu estado, liberando o processador para executar outras tarefas até que receba este sinal. Neste método, o processador

[2] Ponteiros são variáveis que contêm endereços de memória como valores, fazendo referência indireta a um valor específico (DEITEL e DEITEL, 2011).

não precisa ler os registros de estado até que o periférico esteja pronto para transferir dados. Quando o processador recebe o sinal, ele termina o processamento que está realizando (daí o conceito de interrupção) e realiza a transferência de dados. Após a transferência dos dados, o mesmo volta a realizar o processamento interrompido.

Quando a quantidade de dados a transferir for grande, utiliza-se o método de acesso por DMA. Neste caso, transferência é realizada por um circuito específico (não o processador). Após a conclusão da transferência, o processador é notificado através de uma interrupção (STALLINGS, 2010).

Dependendo o método de acesso ao driver, há um aumento do grau de envolvimento do processador, logo é necessário sempre averiguar o método de acesso mais conveniente. No caso da UART2, que é utilizada para depuração do *firmware*, optou-se pela utilização do método programada, pela facilidade de implementação e baixa quantidade de dados transferidos. No Código 2.2 são visualizados os métodos de acesso utilizados.

Código 2.2 – Exemplo de uso do periférico UART2

```
xprintf("IIOT - Serial Debug Testing...");//Camada da Aplicação

(…)//Camada Middleware: Biblioteca Monitor.c manipula a string de caracteres
HAL_UART_Transmit(&huart2, (uint8_t *)&r, 1, 1000);//Camada HAL
```

*Fonte: O AUTOR (2015).*

Para depuração do código, são envolvidas as três camadas do *firmware*: na aplicação é enviado uma string de caracteres (mensagem a ser exibida no computador), após a biblioteca “Monitor.c”, da camada middleware, manipula os caracteres acessando a API “HAL_UART_Transmit” da camada HAL. São necessários quatro parâmetros para sua chamada: ponteiro da estrutura UART, ponteiro para os caracteres, quantidade de caracteres, e tempo de estouro do pooling. Este último é necessário caso demore demasiadamente para o controlador enviar a mensagem, há então um estouro para o controlador abortar o acesso.

## 2.4 Módulo Wi-Fi

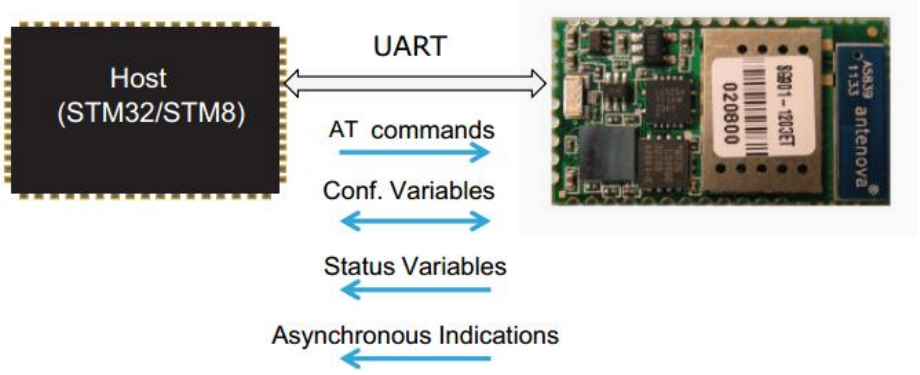
Para comunicação sem fio do coletor, um módulo de Wi-Fi integrado é uma solução muito prática e viável. O módulo de Wi-Fi oferece toda parte de desenvolvimento da pilha TCP/IP, *layout* da antena, blindagem contra interferência eletromagnética e normalização de radiação restrita, já consolidadas e prontas para uso (Wi-Fi, 2015), dando maior velocidade de desenvolvimento ao projeto.

A série de módulos Wi-Fi SPWF01Sx da ST oferece uma solução para integração de dispositivos seguindo o padrão de redes sem fio 802.11 b/g/n. O módulo possui um microcontrolador STM32 ARM Cortex-M3 exclusivo para o trabalho de abstração das camadas complexas da comunicação Wi-Fi. A interface de acesso se torna simples, pois permite a comunicação através de comandos AT via serial-UART.

Por ser focado em aplicações embarcadas, o módulo consome menos que 43mA no modo *Standby*. Possui uma completa pilha TCP/IP integrada, suporte para IPv4 com DHCP (*Dynamic Host Configuration Protocol*), e ainda 16 pinos GPIOs. Um recurso importante é o suporte a conexão de dispositivos via sockets (TCP/ UDP) no modo cliente e servidor, no qual será utilizado para este projeto.

O modelo escolhido neste projeto é o SPWF01SA.11 (Figura 2.6). Ele contém 1.5 MB de memória Flash, permitindo, por exemplo, o carregamento interno de páginas HTML, além de antena de 2.45 GHz ISM embutida.

Figura 2.6: Módulo SPWF01SA.11.

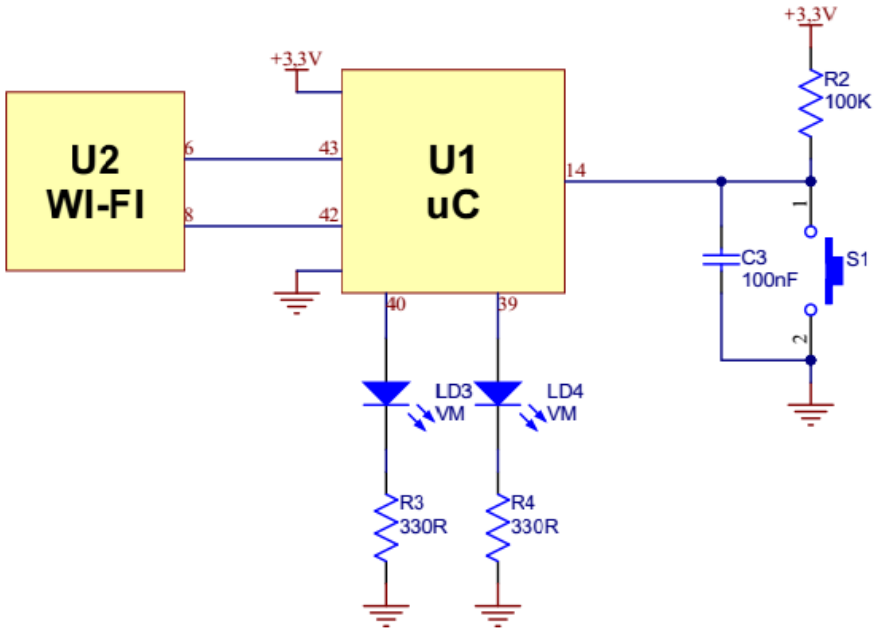



*Fonte: ST (2015).*

### 2.4.1 Configuração do Módulo SPWF01SA.11

A UART1 do microcontrolador foi definida como periférico de comunicação com o módulo Wi-Fi. Os pinos 42 (TX) e 43 (RX) foram configurados para esta função e conectados aos pinos 8 (RXD1) e 6 (TXD1) do módulo Wi-Fi, respectivamente, conforme ilustra a Figura 2.7. A configuração do periférico UART1 no microcontrolador é a seguinte: taxa de transmissão 115200 bits/s; 8 bits de dados; sem paridade; 1 stop bit.

Figura 2.7: Esquemático do SPWF01SA.11 e STM32F0.




*Fonte: O AUTOR (2015).*

Após a configuração inicial da UART1 é preciso estabelecer uma comunicação com o módulo através da interface de comandos “AT full Stack”, a qual é disponibilizada pelo fabricante. Esta API consiste em uma série de comandos, variáveis de configurações, variáveis de status e indicações assíncronas do módulo com o microcontrolador. Na Figura 2.6 mostra estas mensagens e o sentido de envio das mesmas.

Indicações assíncronas podem chegar a qualquer instante (exceto entre comandos) e tem o formato:

*<cr><lf>+WIND:<number>:<descriptive string><cr><lf>.*

Os campos são explicados a seguir:

- <cr>: *carriage return* ou posição inicial, no padrão ASCII: “/r”;
- <lf>: *line feed* ou nova linha, no padrão ASCII: “/n”;
- <*number*>: campo único para cada indicação;
- <*descriptive string*> pode variar conforme a mensagem.

Imediatamente após o reset no módulo, é necessário esperar o recebimento da indicação *“<cr><lf>+WIND:0:Console active<cr><lf>*” para começar a enviar os comandos AT. Os comandos AT seguem sempre o formato:

*AT<cmd><cr>*

*<zero or more response lines>*

*<cr><lf><responsecode><cr><lf>.*

Na primeira linha é enviado pelo microcontrolador o campo “<cmd>” que é único e representa o código do comando, já as duas últimas são a resposta do módulo. A resposta pode ser: “*OK*” ou então “*ERROR: <descriptive text>*”. Para um simples teste de comunicação pode ser enviado/recebido:

*AT<cr>*

*<cr><lf>OK<cr><lf>*

no qual o microcontrolador envia o comando genérico e o módulo caso esteja comunicando responderá com “OK”. No Apêndice A é apresentada a lista completa dos comandos.

Para que o dispositivo coletor e o servidor se comuniquem, os dois devem estar na mesma rede sem fio. Para isto, a tecla de pulsos do kit STM32F0DISCOVERY deve ser pressionada por cinco segundos a fim de configurar o módulo para acesso à rede sem fio 802.11 pré-definida. Nesta configuração o comando “AT+S.SCFG” seta as variáveis de configurações e o comando “AT&W” salva na memória não-volátil do módulo todas as variáveis. Ao dar um reset no módulo, serão acessadas estas variáveis para que o módulo conecte automaticamente na rede sem fio.

## 2.5 Estrutura da Rede

A interface de sockets é o método mais utilizado para uma comunicação transparente via TCP/IP em uma rede, sendo que atualmente existem diversas

aplicações desenvolvidas com esta tecnologia. Neste projeto, um computador é utilizado como servidor, tendo como objetivo abrir um socket e aguardar às conexões. Enquanto o dispositivo coletor e o *tablet* são os clientes, enviando solicitações ao socket servidor para iniciar a conexão (SILVA, 2008).

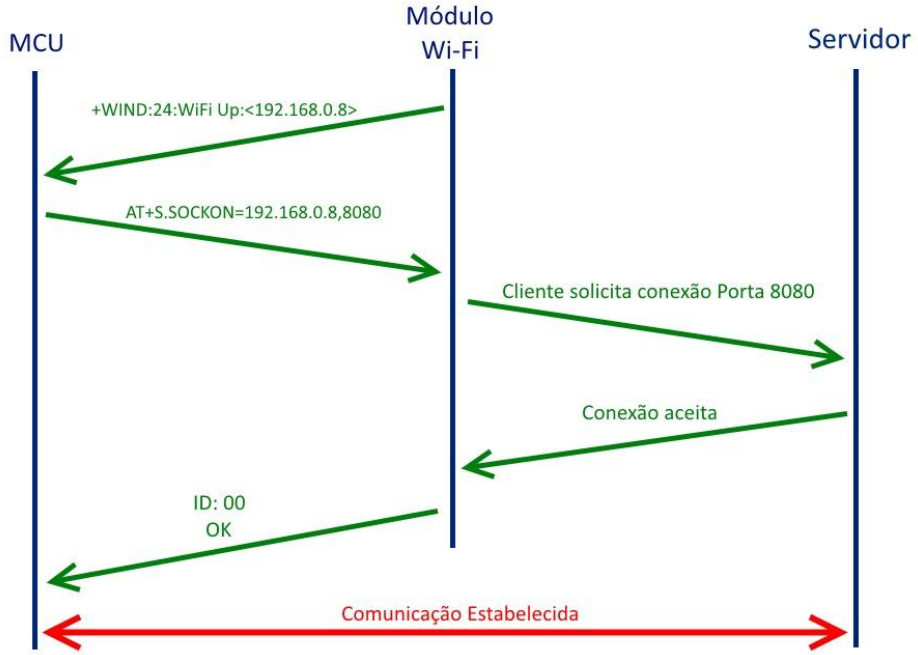
Cada dispositivo da rede possui um endereço IP exclusivo, enquanto as portas representam conexões individuais dentro desse endereço (um computador possui 65535 portas). Ao criar um socket deve-se associar a uma porta específica para que os dados possam ser roteados (SILVA, 2008). O endereço IP e o número da porta do servidor devem ser conhecidos para que os outros dispositivos clientes possam iniciar uma conexão socket.

Os sockets possuem dois modos de operação, baseado no protocolo TCP ou UDP. Serão utilizadas as pilhas de protocolos TCP/IP, em função da maior confiabilidade na entrega de pacotes (Stanford, 2008).

O módulo Wi-Fi do dispositivo coletor deve ser configurado como um co-processador de rede. Isto é, deve tratar e enviar mensagens ao servidor no padrão 802.11, e permitir o acesso facilitado dos dados pelo MCU do coletor. Depois de configurado o módulo Wi-Fi e conectado via Socket os dispositivos, é estabelecido uma comunicação serial entre ambos, logo se faz necessário estabelecer um protocolo de comunicação para troca de mensagens da aplicação. Este será definido na Seção 2.6.

Para uma melhor compreensão do processo de comunicação entre os dispositivos, a Figura 2.8 apresenta um diagrama de sequência simplificado do mesmo.

Figura 2.8: Diagrama de sequência da comunicação Wi-Fi.




*Fonte: O AUTOR (2015).*

Com a conexão Socket servidora já aberta pelo Servidor, é inicializado o dispositivo coletor e aguardado uma mensagem de indicação que o módulo está conectado à rede. Esta mensagem de indicação do Módulo Wi-Fi contém o código

+WIND:24:WiFi Up:< EndereçoIP >,

em que informa sucesso na conexão e o endereço IP. Uma vez conectada à rede, a próxima etapa é solicitar a abertura de um Socket cliente, que é envida pelo MCU com o comando:

AT+S.SOCKON=<EndereçoIP>,<Porta>

no qual indica o endereço IP do servidor e a porta utilizada. Ao chegar esta mensagem no formato TCP/IP ao Servidor, ele irá reservar uma porta exclusiva para a comunicação com o dispositivo coletor e enviar uma resposta ao Módulo Wi-Fi com a confirmação da conexão bem sucedida.

O Módulo Wi-Fi possui até oito conexões Socket cliente disponíveis para acesso, então enviará uma mensagem com o ID disponibilizado para esta comunicação com o Servidor na mensagem:

ID:xx

OK.

A comunicação serial entre o Servidor e o dispositivo coletor foi estabelecida e as mensagens trocadas a partir desta, deverão estar padronizadas através de um protocolo, no qual será apresentado na Seção 2.6.

## 2.6 Protocolo de Comunicação

Esta seção descreve o protocolo de comunicação para a troca de informações entre o servidor, o dispositivo coletor e o *tablet*. Este protocolo é encapsulado dentro do protocolo TCP/IP, qual já contém uma série de métodos para garantia de entrega de pacotes e detecção de erros (STALLINGS, 2010). Então se faz necessário um protocolo simples, porém eficaz, para maior velocidade na troca de dados da aplicação.

Tendo base no protocolo de comunicação ModBus RTU (*Remote Terminal Unit*), o qual possui uma menor densidade de caracteres por mensagem, aumentando o desempenho (Alfa Instrumentos, 2000), foi definido o protocolo de comunicação apresentada na Tabela 2.1.

Tabela 2.1: Protocolo de comunicação da aplicação.

| **Byte inicial:** | 0 | 1.0 | 1.3 | 2 | 3 | 4 | 5-6 | 7-n | (n+1) |
|---|---|---|---|---|---|---|---|---|---|
| **Código:** | STX | TYP | RES | DST | SCR | CMD | DSZ | Data | CCS |
| **Tamanho:** | 1 byte | 4 bits | 4 bits | 1 byte | 1 byte | 1 byte | 2 byte | n byte | 1 byte |

| **Código** | **Descrição** |
|---|---|
| STX (Start of Transmission) | Início de transmissão: Código 0x02 |
| TYP (Type of protocol) | Tipo do protocolo: Código 0x1 |
| RES (Reserved) | Reservado: Código 0x0 |
| DST (Destination Address) | Endereço de destino (0xFF = Broadcast) |
| SCR (Source Address) | Endereço de origem |
| CMD (Command Code) | Código do comando da mensagem |
| DSZ (Data Size) | Tamanho dos dados |
| DATA | Dados |
| CCS (Checksum) | CheckSum: Soma dos bytes entre STX e CCS |

*Fonte: O AUTOR (2015).*

O primeiro byte do protocolo, STX, indica o início de transmissão. Logo após o tipo do protocolo (neste caso “1”) serve para diferenciar em caso de futuras alterações no formato deste protocolo. Os campos DST e SCR indicam o endereço de destino e o endereço de origem da mensagem, respectivamente, logo há 255 endereços disponíveis neste protocolo. Caso for necessária uma mensagem global, coloca-se o campo DST em “0xFF”.

O campo CMD é um dos mais importantes, pois é um código único qual identifica a informação que a tomada ou o dispositivo móvel está enviando na mensagem. Conforme o comando enviado, requer também o envio dos dados, o campo DSZ indica a quantidade de bytes do campo DATA. Caso for zero, não é necessário dados na mensagem. Por fim, há o campo CCS, que faz a soma em uma variável de 8 bits (quando estoura reinicia contagem) dos bytes entre os campos STX e CCS, a fim de detectar erros na mensagem.

### 2.6.1 Lista de Comandos

Nesta seção serão detalhados os comandos mais importantes da comunicação. Na Tabela 2.2 é demonstrada a lista atual dos comandos utilizados no campo CMD do protocolo. A lista pode ter no máximo de 255 comandos.

Tabela 2.2: Lista de códigos dos comandos.

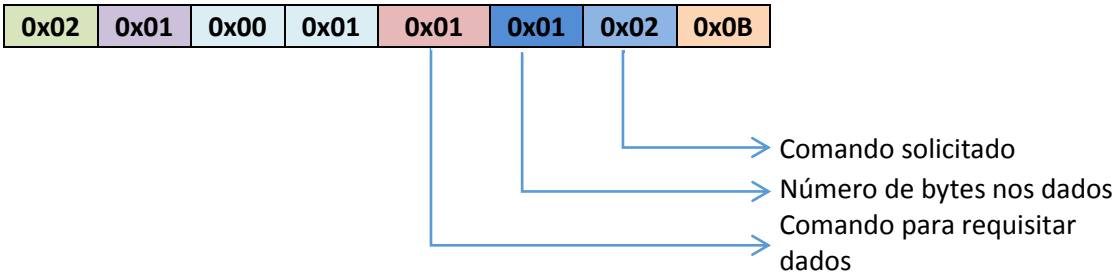
| Código | Descrição | Tamanho dos dados | Quem envia |
|---|---|---|---|
| 0x01 | Requisitar comandos | 1 | todos |
| 0x02 | Versão de *firmware* | 3 | coletor/*tablet* |
| 0x03 | ID do dispositivo | 6 | coletor/*tablet* |
| 0x04 | Informações da OP (ordem de produção) | 26 | servidor |
| 0x05 | Iniciar OP | 2 | *tablet* |
| 0x06 | Informações da peça | 3 | coletor |
| 0x07 | Informações das máquinas | 2 | servidor |

*Fonte: O AUTOR (2015).*

Quando algum dispositivo quer receber uma mensagem que outro dispositivo é responsável por enviar, o mesmo encaminha uma mensagem com código 0x01. Além deste código, no campo de dados da mensagem deve conter o código qual deseja receber. Na Figura 2.9 é apresentado um exemplo da utilização deste

comando, no qual o servidor solicita ao dispositivo coletor o envio da versão de *firmware* (Código 0x02).

Figura 2.9: Exemplo comando 0x01.




*Fonte: O AUTOR (2015).*

Uma mensagem com comando 0x03 é enviada pelo coletor e pelo *tablet* ao servidor para informar a identidade (ID) do dispositivo, a fim de posteriormente identificar a máquina. Cada dispositivo ao entrar na rede deve possuir um ID único, no qual o servidor irá utilizar para registrar os dados de cada máquina corretamente.

O comando 0x06 contém o número de peças produzidas e o status da máquina, também é enviado do dispositivo coletor para o servidor, e o servidor repassa a mesma mensagem ao *tablet* correspondente à máquina. A Tabela 2.3 apresenta o campo dos dados para esta mensagem.

Tabela 2.3: Dados do comando Informações da peça (0x06).

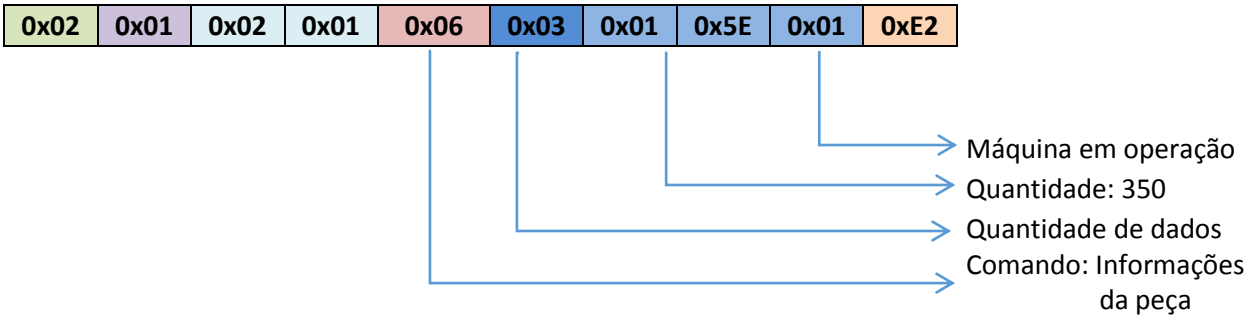
| Byte dos dados | Nome | Descrição |
|---|---|---|
| 0 | Número de peças | High |
| 1 | | Low |
| 2 | Status da máquina | 0= parada; 1=em operação |

*Fonte: O AUTOR (2015).*

São necessários dois bytes no campo dos dados do número de peças, já que o tamanho da variável tem 16 bits. Como este protocolo segue o padrão Big endian[3] é enviado o byte mais significativo antes. Na Figura 2.10 é demonstrado um exemplo quando a quantidade é 350 peças (0x015E).

[3] A definição do formato Big endian diz que o byte mais significativo de qualquer campo de múltiplos bytes é armazenado no menor endereço de memória. Assim, o byte mais significativo é enviado antes. Protocolos como TCP/IP e máquinas virtuais Java seguem este mesmo padrão (BARR e BROWN, 2007).

Figura 2.10: Exemplo comando 0x06.




*Fonte: O AUTOR (2015).*

A mensagem do comando 0x04 contém informações relacionadas a uma ordem de produção (OP), como o número da ordem, data da emissão, data da entrega, código e descrição do produto. Quando o operador começar a produzir, o seguinte processo é executado: o operador digita o número da OP no *tablet*, será solicitado o enviado da mensagem 0x05 (através da mensagem 0x01 – Requisitar comandos). O servidor por sua vez busca as ordens de produção no banco de dados, caso a OP for encontrada é enviada a mensagem 0x05 com os dados. A Tabela 2.4 apresenta o campo dos dados para esta mensagem.

Tabela 2.4: Dados do comando Informações da OP (0x04).

| Byte dos dados | Nome | Descrição |
|---|---|---|
| 0-1 | Número da OP | Big-endian |
| 2-3 | Quantidade da OP | Big-endian |
| 4 | Data de emissão | dia 1-31 |
| 5 | | mês 1-12 |
| 6 | | ano 0-255 |
| 7 | Data de entrega | dia 1-31 |
| 8 | | mês 1-12 |
| 9 | | ano 0-255 |
| 10-15 | Código da peça | string ascii |
| 16-25 | Nome da peça | string ascii |

*Fonte: O AUTOR (2015).*

O comando 0x05 é enviado pelo *tablet* para indicar que a OP que estava apenas sendo visualizada, teve início do processo de produção. O servidor ao recebê-la envia também ao dispositivo coletor, preparando o coletor para obter informações da máquina.

## 2.7 Aplicativo para Dispositivos Android

Android é uma pilha de *software* de código aberto para uma ampla variedade de dispositivos móveis. Este sistema operacional móvel é liderado pela empresa Google, que provê diversas ferramentas e documentos para desenvolvedores de aplicativos (Android Source, 2015).

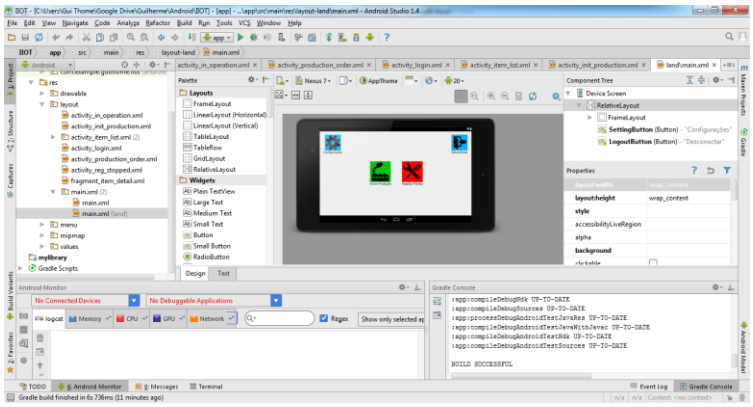
Esta seção apresenta a programação e a interface de programação do aplicativo do *tablet* para SO Android. A partir daqui serão apresentados códigos em linguagem de programação em Java[4], assim como *layouts* e definições da interface desenvolvida para dispositivos móveis Android.

### 2.7.1 Plataforma de Desenvolvimento

Utilizou-se neste projeto o Android Studio – IDE de programação para plataforma Android. Este ambiente de desenvolvimento possui um sistema de compilação Gradle para construção de compilações customizadas e suporte a múltiplos APKs (Android Packages). Integrado à IDE existe o kit de ferramentas SDK (*Software* Development Kit), que inclui um emulador, interface gráfica, depurador, entre outras ferramentas (Android Source, 2015).

A linguagem de programação para Android é o Java, desta forma, é necessário também a instalação do JDK 7 (Java Development Kit) ou superior. Na Figura 2.11 é ilustrada a interface de programação do Android Studio.

Figura 2.11: IDE Android Studio.



*Fonte: O AUTOR (2015).*

[4] http://www.java.com/

### 2.7.2 *Layout* principal do Aplicativo

O Android tem diversos níveis de API para configurar a compatibilidade dos aplicativos com múltiplas versões do sistema operacional. Níveis de API menores são compatíveis com mais dispositivos, porém tem menos recursos disponíveis. Neste projeto optou-se pela versão mínima SDK para Android 4.0 “IceCreamSandwich” (nível de API 14) que detêm compatibilidade com 90,4% dos dispositivos no mercado (Android Dashboard, 2015). Além do nível mínimo da API, é necessário especificar o nível de API que o aplicativo destina-se a rodar. Neste aplicativo foi utilizado a API de nível 21 (Android 5.0 “Lollipop”).

No *layout* do aplicativo buscou-se criar a interface intuitiva e de fácil adaptação ao operador, deste modo foi criado uma *Activity*[5] com botões e imagens maiores. Também foi bloqueado o giro automático a tela, deixando sempre na orientação horizontal, denominada como Landscape (Android Developer, 2015).

Figura 2.12: *Activity* principal do aplicativo.


*Fonte: O AUTOR (2015).*

Foram disponibilizados no *layout* principal três botões com as seguintes descrições: Iniciar Produção, Registrar Parada, Configurações. O botão Iniciar Produção é utilizado para começar a produção através de uma OP (ordem de produção). O botão Registrar Parada é necessário para indicar alguma parada de máquina, indicando posteriormente o motivo do ocorrido. O botão Configurações

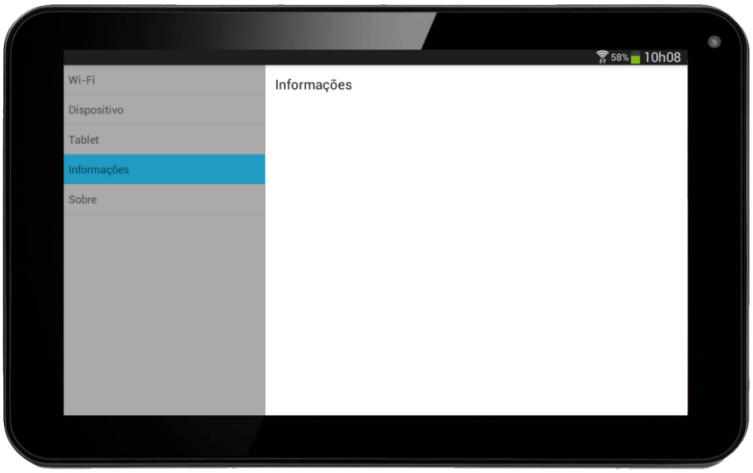
[5] *Activity* é uma classe gerenciadora de UI (Interface com o usuário). Quase todas as *Activities* interagem com o usuário (Android Developer, 2015).

possui parâmetros para configurar a rede Wi-Fi, alterar parâmetros do dispositivo coletor, informações do aplicativo, dentre outros.

Para definir o *layout* principal deve ser criado um arquivo XML (main.xml), no qual contém três botões (classe Button). Pode-se configurar a imagem de fundo, cor e espessura das letras e o ID destes botões. O Apêndice C.1 apresenta o código em XML deste *layout* e o Apêndice C.2 apresenta a programação em linguagem Java.

Na tela de configurações foi criada no lado esquerdo da tela uma lista dos grupos de configurações (Wi-Fi, Dispositivo Coletor, *Tablet*, Informações), e no lado direito pode-se selecionar e modificar os parâmetros de cada grupo. O procedimento fica reunido em apenas uma tela, tornando o sistema mais prático. O nome desta interface, que geralmente é utilizada em *tablets*, é Fragments (fragmentos de uma tela). No lado esquerdo é criado um ListView, que exibe uma lista de itens roláveis, onde o usuário pode selecionar um item, que por sua vez será exibido no lado direito através de Fragments. A Figura 2.13 apresenta esta *Activity*.

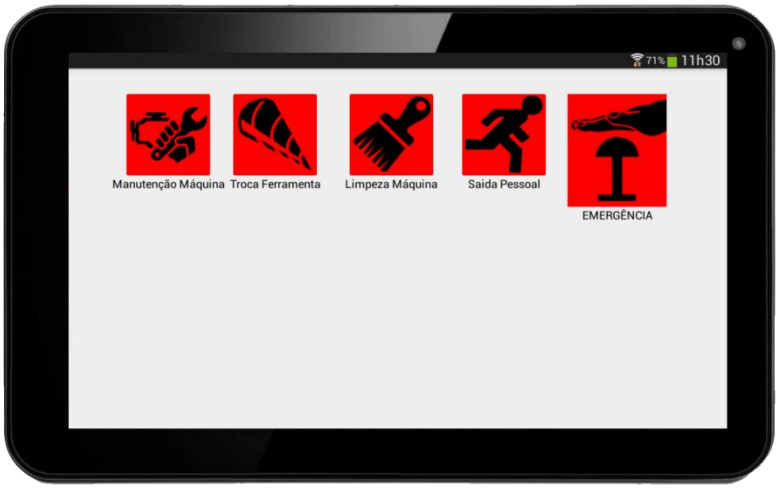
Figura 2.13: *Activity* de configurações.




*Fonte: O AUTOR (2015).*

O botão Registrar Parada grava no sistema uma eventual parada de máquina, com isso o operador é direcionado para uma próxima tela para informar o motivo da parada. Nesta *Activity* foram criados outros cinco botões para o operador selecionar: manutenção da máquina, limpeza da máquina, troca de ferramenta, saída pessoal ou emergência, conforme ilustrado na Figura 2.14.

Figura 2.14: *Activity* de registro de parada.




*Fonte: O AUTOR (2015).*

O botão intitulado Iniciar Produção irá habilitar o funcionamento da máquina. No evento do clique, o usuário é direcionado a outra *Activity* para digitar o número da OP que será iniciada, conforme é ilustrado na Figura 2.15. Esta tela deve possuir um TextEdit (campo para digitação do usuário) com formato de entrada numérico e um botão de confirmação. Após a confirmação, o evento setOnClickListener do botão será encarregado de criar uma mensagem de requisição (código 0x01 do protocolo) a ser entregue ao servidor, a fim de solicitar informações sobre a OP (código 0x04). Se ela estiver cadastrada, o servidor irá devolver a mensagem com as informações da OP, caso contrário é informado o erro encontrado.

Figura 2.15: *Activity* de digitação da OP.


*Fonte: O AUTOR (2015).*

Ao receber uma mensagem do servidor com as informações da OP, é criada uma nova *Activity* para a visualização destas informações. O usuário irá conferir se digitou o número da OP corretamente e clicar no botão Iniciar, no qual é criada a *Activity* InOperation.xml. Esta é a tela em que indica o início de produção de uma OP pelo operador, automaticamente será enviada uma mensagem indicando esta operação (código 0x05). O servidor irá repassar a mensagem ao coletor para que seja zerado o contador de peças, assim o coletor a cada peça produzida enviará uma mensagem com a quantidade em tempo real produzida (código 0x06).

Figura 2.16: *Activity* InOperation.


*Fonte: O AUTOR (2015).*

A Figura 2.16 ilustra a *Activity* InOperation.java, que além de possuir as mesmas informações da OP, tem dois botões para registrar uma parada e encerrar a OP. Também possui um contador progressivo que indica o tempo de produção (classe Chronometer.java), e um ProgressBar que é uma barra que indica o progresso das quantidades produzidas. A borda na cor verde indica que a máquina está em produção, porém quando registrado uma parada, a mesma fica vermelha.

## 2.8 Servidor

O servidor é o principal componente da estrutura proposta neste trabalho, pois é responsável por se comunicar individualmente com os dispositivos das máquinas (coletor e *tablet*), e ainda permitir o acesso dos dispositivos portáteis e computadores dos gestores em um navegador para internet. Deu-se o nome de servidor, porém a proposta é desenvolver um *software* que pode ser rodado em

qualquer computador pessoal. Em empresas é recomendado ficar hospedado em um servidor (Linux, Windows Server, etc), qual se encontra na maior parte do tempo *online* e pode ser acessado a qualquer momento.

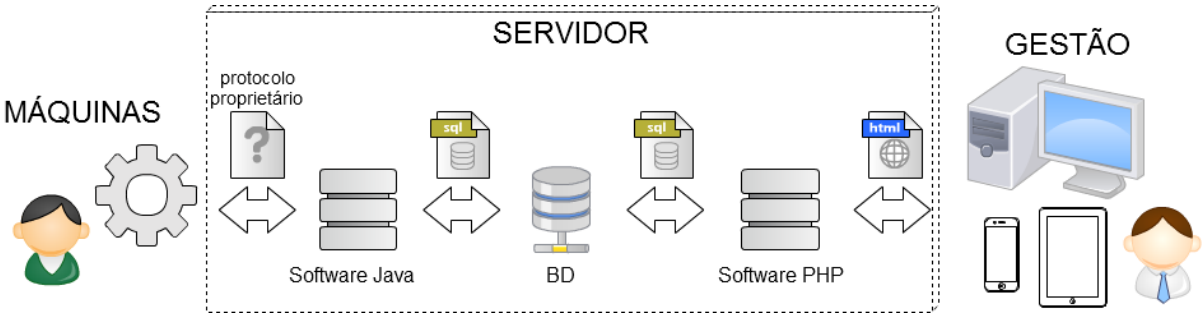
No servidor rodará diferentes softwares simultaneamente para atender ambos os lados da comunicação. A comunicação com as máquinas possui uma interface TCP/Socket, que coleta dados das máquinas e armazena em um Banco de Dados (BD). Este *software* recebeu o nome de Software Java, que como o nome já diz é feito em linguagem Java. Ele é responsável pela troca de informações via Socket do servidor com as máquinas, utilizando o protocolo proprietário (citado na Seção 2.6).

O outro *software* foi nomeado *Software* PHP, feito em linguagem PHP, foi desenvolvido para a comunicação com os *desktops* e dispositivos portáteis dos gestores. Estes dispositivos enviam requisições no protocolo HTTP (*Hypertext Transfer Protocol*), as informações são processadas no BD e retornadas via páginas HTML que são visualizadas através de um navegador.

O banco é a interface entre ambas as aplicações, fazendo a escrita e ao mesmo tempo processamento os dados para consultas dos dispositivos dos gestores (outro lado da comunicação).

Como é demonstrada na Figura 2.17, a arquitetura desenvolvida possui dois *softwares* acessando simultaneamente o BD. O banco é a interface entre ambas as aplicações, fazendo a escrita e ao mesmo tempo processamento os dados para consultas dos dispositivos dos gestores (outro lado da comunicação).

Figura 2.17: Arquitetura de *softwares* do Servidor




*Fonte: O AUTOR (2015).*

Na Subseção 2.8.1 será explicado sobre o banco de dados. Na Subseção 2.8.2 será exposto a respeito do *Software* Java (comunicação com as máquinas), e na Subseção 2.8.3 será explanado o *Software* PHP (comunicação com os gestores).

### 2.8.1 Banco de Dados

Um Banco de Dados (BD) é utilizado para armazenar os dados das máquinas e operadores e atender as requisições de consultas dos dispositivos dos gestores. Dentre as diversas vantagens em utilizar um sistema de Banco de Dados pode-se citar as principais como: compartilhamento dos dados, segurança, respostas rápidas às solicitações e flexibilidade em função da independência entre os dados e os programas (MEDEIROS, 2013).

A linguagem padrão para manipular bancos de dados relacionais através dos SGBDs (Sistema de Gerenciamento de Banco de Dados) é a SQL (*Linguagem de Consulta Estruturada*). Esta linguagem é usada para interação com o SGBD a fim de executar tarefas como busca, inserção e alteração de registros, criação de tabelas de dados, entre outras (MEDEIROS, 2013).

Neste projeto foi utilizado o MySQL[6], que utiliza a linguagem SQL e é um dos mais populares SGBDs do mercado. Suas principais vantagens são o fato de rodar em praticamente qualquer sistema operacional e ser um *software* livre (qualquer um pode estudá-lo ou alterá-lo conforme a necessidade). Seus dados são armazenados em tabelas constituídas de colunas e linhas, que é uma coleção de dados relacionais.

Para este desenvolvimento foi utilizado o *software* Wamp Server, que é um pacote de instalação do servidor web Apache, linguagem de programação PHP e bando de dados MySQL. O *software* possui uma interface na barra do Windows no qual é possível iniciar os serviços necessários para rodar o banco de dados e também o servidor Apache, que será utilizado para as requisições HTTP (explicado na Subsecção 2.8.3).

Neste projeto foram criadas algumas tabelas, dentre elas pode-se destacar a Tabela 2.5 e Tabela 2.6.

[6] http://www.mysql.com

Tabela 2.5: Banco de Dados – Tabela de OPs

| ID | Status | Número | Código da peça | Nome da peça | Qtd | Qtd produzida | Data de emissão | Data de entrega |
|---|---|---|---|---|---|---|---|---|
| 1 | EM ANDAMENTO | 12345 | 123.456 | Polia | 12 | 2 | 06/11/2015 | 08/11/2015 |
| 2 | EM ANDAMENTO | 12346 | 123.490 | Arruela | 100 | 35 | 07/11/2015 | 09/11/2015 |
| 3 | CONCLUIDA | 12347 | 123.551 | Parafuso | 60 | 60 | 08/11/2015 | 10/11/2015 |

*Fonte: O AUTOR (2015).*

Tabela 2.6: Banco de Dados – Tabela de Máquinas

| ID | Nome | ID Coletor | ID *Tablet* | Status | Número OP |
|---|---|---|---|---|---|
| 1 | SER002 | 1 | 3 | PARADA | |
| 2 | CNC001 | 3 | 1 | PRODUZINDO | 12346 |
| 3 | FRZ017 | 2 | 2 | SAIDA PESSOAL | 12347 |

*Fonte: O AUTOR (2015).*

A Tabela 2.5 e a Tabela 2.6 possuem exemplos aleatórios a fim de demonstrar os tipos de dados que podem ser inseridos e editados. Ambas as tabelas possuem a primeira coluna destinada ao identificador, que é configurado como auto incremental a fim de diferenciar de maneira segura as linhas da tabela. Ao inserir, por exemplo, uma nova OP no sistema, devem ser preenchidos as oito colunas de dados (exceto o ID). Os dados das linhas sofrerão leituras e escritas no decorrer da sua vida, como consultas através do campo “Número” e alterações do campo “Status”.

A tabela das máquinas, além do campo ID, possui um nome que serve para identificar a máquina pelos usuários da empresa. As colunas “ID Coletor” e “ID *Tablet*” registram o identificador do dispositivo que está conectado à maquina. Também é possível alterar em tempo real o status da máquina e o número da OP. O código completo da classe DataBase.java é apresentado no Apêndice C.3.

Será explicado posteriormente como se dá o acesso às tabelas do DB. A Subseção 2.8.2.1 apresentará a API utilizada para acesso ao banco utilizando linguagem Java, enquanto a Subseção 2.8.3.1 demonstra o acesso programando em linguagem PHP.

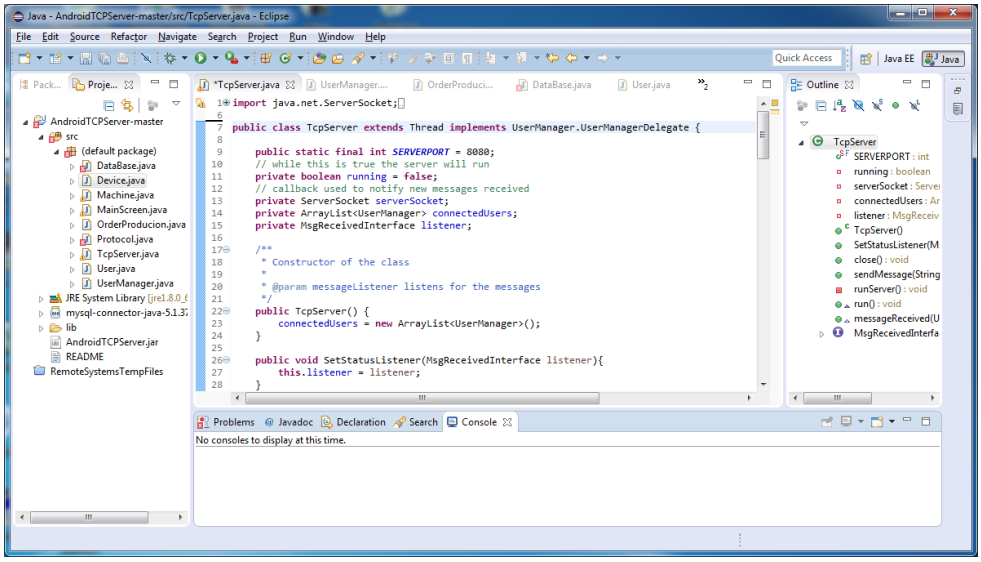
### 2.8.2 *Software* Java

Para a comunicação com os *tablets* e dispositivos coletores das máquinas foi desenvolvido um *software* utilizando a linguagem de programação Java e o ambiente de desenvolvimento Eclipse[7]. Foi escolhida esta plataforma devido a ser uma tecnologia gratuita e possuir portabilidade, já que não requer alteração do código Java para rodar em outras plataformas de sistemas operacionais, facilitando assim a migração.

Outro fator muito importante é que o aplicativo Android dos *tablets* também é desenvolvido em linguagem Java, assim é possível reaproveitar uma parte do código. Além de possuir uma grande quantidade de materiais e exemplos disponíveis para consulta, por ser uma linguagem amplamente utilizada por desenvolvedores de aplicações em rede.

O Eclipse é a IDE Java mais utilizada no mundo e tem seu código livre. Apesar de ser escrito em Java, a biblioteca gráfica usada no Eclipse usa componentes nativos do sistema operacional. Ele possui forte orientação ao desenvolvimento baseado em plug-ins para emular o desenvolvimento da plataforma e compreender vários tipos de linguagens diferentes. Na Figura 2.18 é demonstrado o Eclipse – ambiente de desenvolvimento deste projeto.

Figura 2.18: IDE Eclipse




*Fonte: O AUTOR (2015).*

[7] http://eclipse.org

O *Software* Java deve permitir a abertura de uma conexão servidora TCP através de Sockets, no qual o servidor deve aguardar solicitações de conexões pelos clientes em uma porta pré-definida. Para isso foi importado o pacote “java.net.ServerSocket” que permite a criação da classe TcpServer, a qual abstrai os detalhes da camada TCP/IP.

Código 2.3 – Trecho de código da classe TcpServer

```
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;

public class TcpServer extends Thread implements UserManager.UserManagerDelegate {

   public static final int SERVERPORT = 8080;
   private ServerSocket serverSocket;
   private ArrayList<UserManager> connectedUsers;

   public TcpServer() {
      connectedUsers = new ArrayList<UserManager>();
   }
```

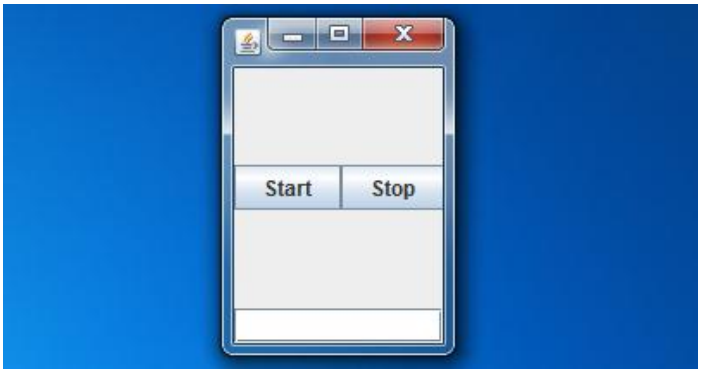
*Fonte: O AUTOR (2015).*

No trecho de Código 2.3 é apresentado a classe TcpServer, que é a interface entre as classes ServerSocket, para a conexão Socket, e a classe UserManager, para gerenciar os usuários. A primeira classe importante é a ServerSocket na qual é responsável por esperar a conexão dos clientes. Esta classe possui um construtor para passar a porta usada na escuta das conexões, neste caso utilizou-se a 8080.

A classe UserManager é responsável por fazer o gerenciamento dos usuários (portas) conectados. Como o servidor recebe conexões de diversos dispositivos diferentes, é necessário identificar cada um e guardar em um ArrayList. Cada dispositivo armazena os dados em um objeto denominado User, e tem as seguintes informações: número da porta, ID e nome. Tais informações são recebidas através da mensagem com código 0x03 do protocolo (ID do dispositivo). O código completo das classes UserManager.java e TcpServer.java é demonstrado nos Apêndices D.1 e D.2, respectivamente.

A interface do *software* é extremamente simples. Na tela serão exibidos dois botões: Start e Stop. O botão Start inicia a comunicação Socket Server, ou seja, abre a porta de comunicação definida como 8080 e aguarda as requisições de conexão dos *tablets* e coletores. O botão Stop fecha a porta de comunicação. A Figura 2.19 representa a tela do *software* em Java.

Figura 2.19: Representação do *software* em Java.

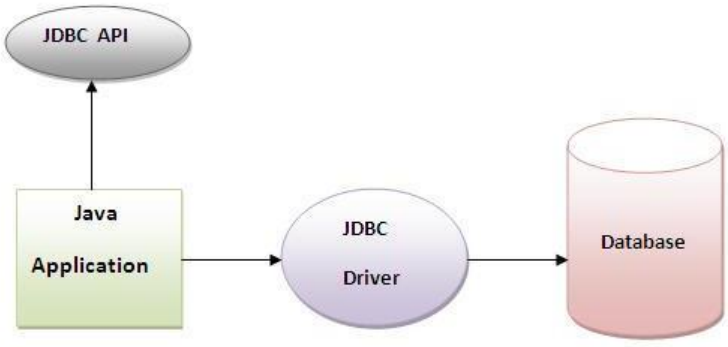



*Fonte: O AUTOR (2015).*

### 2.8.2.1 *Acesso ao banco de dados*

Na linguagem Java, existe a API chamada JDBC (*Java Database Connectivity*), que possui um conjunto de classes e interfaces para conectar-se ao banco de dados e executar instruções através da linguagem SQL. Esta API utiliza drivers para se conectar com qualquer tipo de banco de dados relacional. Cada SGBD deve possuir um driver para converter o protocolo JDBC da aplicação para o protocolo proprietário que o banco de dados utiliza. A Figura 2.20 demonstra a estrutura da API JDBC (JavaPoint, 2015).

Figura 2.20: Estrutura JDBC




*Fonte: JavaPoint (2015).*

Foi criada uma classe chamada *DataBase* que referencia o pacote java.sql, o mesmo tem as classes e interfaces necessárias para o funcionamento do banco de dados, como o *DriverManager, Connection, Statement, ResultSet*, entre outras. Por exemplo, a classe *DriverManager* registra o driver MySQL (BD utilizado neste trabalho) para poder oferecer os métodos estáticos de gerenciamento do driver

JDBC. O trecho de código Código 2.4 apresenta a parte inicial da classe *DataBase*, que foi criada para reunir as interfaces da API JDBC, e com isso criar as tabelas.

Código 2.4 – Trecho de código da classe *DataBase*

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public final class DataBase {
  private Connection connect = null;
  private Statement statement = null;
  private PreparedStatement preparedStatement = null;
  private ResultSet resultSet = null;

  public void connectDB() throws Exception {
  Class.forName("com.mysql.jdbc.Driver");  // Carrega o driver MySQL
  connect = DriverManager.getConnection("jdbc:mysql://localhost/database?"
          + "user=root");  // Instala a conexão com o BD
}
```

*Fonte: O AUTOR (2015).*

A classe *DriverManager* é responsável por se comunicar com todos os drivers disponíveis. Para isso, é invocado o método estático getConnection com uma String que indica qual banco a ser conectado. Essa String – chamada de String de conexão JDBC – será utilizada para acesso ao banco da seguinte forma:

jdbc:mysql://ip/nome_do_banco?user=nome_do_usuário

onde o campo “ip” representa o IP da máquina do servidor; “nome_do_banco” é o nome do BD a ser utilizado; “nome_do_usuário” representa o usuário cadastrado no BD. O código completo da classe *DataBase* é visualizado no Apêndice D.3.

### 2.8.3 *Software* PHP

Para a interface de visualização e controle das máquinas foi necessário desenvolver um servidor web, este recebeu o nome de *Software* PHP. É um programa de computador responsável por receber pedidos HTTP de clientes e responder as informações solicitadas através de uma página HTML visualizadas em um navegador. A principal vantagem desta tecnologia é que diferentes tipos de dispositivos (*tablets*, *smartphones* e *desktops* dos gestores) de diferentes marcas

poderão acessar as informações necessitando possuir apenas uma interface de rede.

A linguagem HTML possui certas limitações. Por exemplo, chegando uma requisição HTTP para o servidor, ele simplesmente não envia uma página HTML pronta. O servidor deve acessar o BD e gerar a página HTML de forma dinâmica com as informações desejadas. Para isso foi utilizada a linguagem PHP, que possui mais recursos (Caelum, 2015), em conjunto com o servidor Apache (ambos instalados pelo Wamp Server).

O servidor web Apache é um dos mais populares servidores livres do mundo. Alguns dos motivos são sua excelente performance, segurança, compatibilidade com diversos sistemas operacionais. Sua aplicação mais conhecida é a que combina o servidor Apache com a linguagem PHP e o banco de dados MySQL (Infowester, 2015), exatamente a mesma utilizada neste projeto.

Para agilizar o desenvolvimento das páginas HTML, utilizou-se um framework chamado Bootstrap[8], que contém interfaces para criação de forma simplificada de páginas HTML. Uma das principais vantagens é a sua funcionalidade responsiva, ou seja, ajusta as páginas conforme as dimensões da tela do usuário. Assim, a aplicação se adapta facilmente a um *desktop* tanto quanto a um dispositivo portátil.

O trecho do Código 2.5 demonstra a estrutura utilizada para desenvolver a página HTML que será gerada nas requisições dos clientes. O campo “html lang” define o idioma Português Brasileiro. O campo “head” é o cabeçalho, e contém algumas definições para importação do framework Bootstrap. O campo “body” contém o código para execução da página, inclusive campos em linguagem PHP (foi removido deste trecho de código). O Apêndice E.1 contém o código completo deste *software*.

[8] http://getbootstrap.com/

Código 2.5 – Trecho de código do arquivo index.php.

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta http-equiv="Content-Type" content="text/html"; charset=iso-8859-1">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap core CSS -->
    <link href="assets/dist/css/bootstrap.min.css" rel="stylesheet">

    <!-- Custom styles for this template -->
    <link href="assets/docs/examples/justified-nav/justified-nav.css" rel="stylesheet">

    <!-- Just for debugging purposes. Don't actually copy these 2 lines! -->
    <script src="assets/js/ie-emulation-modes-warning.js"></script>

  </head>
  </body>
     <!-- Código... -->
  </body>
</html>
```

*Fonte: O AUTOR (2015).*

### 2.8.3.1 *Acesso ao banco de dados*

O acesso ao banco de dados MySQL é feito através de funções nativas da linguagem PHP. Para conectar ao BD são utilizadas as funções: “mysqli_connect()” e “mysqli_select_db()”. O Código 2.6 demonstra como é realizada esta conexão.

Código 2.6 – Conexão com BD em PHP

```
$conexao = mysqli_connect("localhost","root");
if (!$conexao)
       die ("Erro de conexão com localhost);

$banco = mysqli_select_db($conexao,"database");
if (!$banco)
       die ("Erro de conexão com banco de dados);
```

*Fonte: O AUTOR (2015).*

No Código 2.6, a função “mysqli_connect” necessita dos parâmetros do IP do banco de dados e o usuário. A variável “conexao” recebe os parâmetros da conexão com o BD, e posteriormente é repassada nos parâmetros da função “mysqli_select_db”, juntamente com o nome do BD. Caso as variáveis retornem nulas, uma mensagem de erro será mostrada.

Para consultas nas tabelas é utilizada a função “mysqli_query”, e como parâmetros são passadas a conexão e uma String (no padrão de linguagem SQL)

com o comando “SELECT”. No Código 2.7 é demonstrado um trecho do código que contém a consulta de uma OP.

Código 2.7 – Consulta ao BD em PHP

```
$query = "SELECT status,code FROM op WHERE id = " . + $idUser;
$result = mysqli_query($conexao,$query); // run the query and get the result object.
if (!$result) { // check for errors.
  echo 'Could not run query: ' . mysql_error();
  exit;
}
```

*Fonte: O AUTOR (2015).*

O comando “SELECT” irá selecionar as colunas “status” e “code” da tabela “op” no qual o campo “id” é igual a um valor informado pelo usuário (quando clicar sobre a aba da tabela referente a maquina desejada). Outros comandos como o “INSERT”, “DELETE” e “UPDATE” podem ser utilizados para incluir, remover e atualizar os dados na tabela selecionada. O código completo em PHP pode ser visto no Apêndice E.1.

# 3 RESULTADOS FINAIS

Neste capítulo serão apresentados os testes e resultados da operação em tempo real com as três partes do sistema: o *tablet* e o coletor para interação com a máquina, o servidor para armazenar e processar os dados, e os dispositivos para visualizar as informações. Todos os componentes serão conectados em uma rede através de um roteador Wi-Fi.

Na Seção 3.1 foi elaborada uma tabela com especificações dos requisitos necessários para avaliação do sistema proposto. A Seção 3.2 apresenta a estrutura utilizada nos testes e os resultados finais obtidos nos testes.

## 3.1 Requisitos de Teste

Antes da realização dos testes do sistema, foi elaborada uma tabela que contém os Requisitos Funcionais (RF) e os Requisitos Não Funcionais (RNF) que serão avaliados. Os requisitos funcionais referem-se às funções e serviços que o sistema deve fornecer. Os requisitos não funcionais são as qualidades e restrições do sistema. A Tabela 3.1 apresenta uma lista dos RF e RNF que serão avaliados nos testes.

Tabela 3.1: Requisitos do sistema.

| Código | Descrição |
|---|---|
| RF001 | Buscar no banco de dados uma OP |
| RF002 | Mostrar informações da OP |
| RF003 | Registrar parada de máquina |
| RF004 | Tela de configurações do sistema |
| RF005 | Interface gráfica para acompanhamento da OP em produção |
| RF006 | A tela de acompanhamento da OP deve ter cronômetro e barra de progresso das peças |
| RF007 | Entrada de contagem de peças produzidas. |
| RF008 | Saída para habilitar o funcionamento da máquina |
| RF009 | Interface para encerramento da OP |
| RF010 | Disponibilizar status das máquinas aos gestores |
| RF011 | Disponibilizar número da OP das máquinas em produção aos gestores |
| RNF001 | Os equipamentos devem ser interligados em uma rede Wi-Fi |
| RNF002 | Hardware Wi-Fi para comunicação com a máquina |
| RNF003 | IHM Wi-Fi para comunicação com o operador |
| RNF004 | Servidor para comunicação com dispositivos móveis |
| RNF005 | Servidor deve possuir um banco de dados |

| Código | Descrição |
|---|---|
| RNF006 | Acesso dos gestores será através de navegadores de internet de qualquer dispositivo portátil ou *desktop*. |
| RNF007 | Tempo de busca e atualização da OP no *tablet* deve ser inferior a 1 segundo |
| RNF008 | Tempo de atualização das informações para os gestores deve ser inferior a 10 segundos |

*Fonte: O AUTOR (2015).*

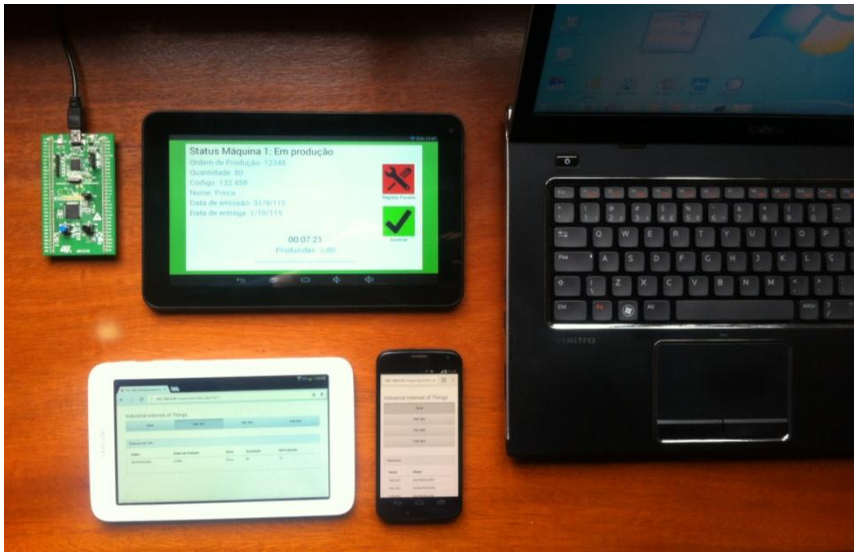
## 3.2 Estrutura de Testes e Resultados

A estrutura de rede construída para os testes possui os seguintes equipamentos:

- 1 dispositivo coletor;
- 1 *tablet* Multilaser M9 para interface da máquina;
- 1 roteador D-Link WBR-2310
- 1 *notebook* Dell – Windows 7 utilizado como Servidor;
- 1 *tablet* Samsung Galaxy Tab tela 7” utilizado pelo gestor;
- 1 *smartphone* Motorola tela 4,7” utilizado pelo gestor;
- 1 *notebook* Sansung – tela 15” utilizado pelo gestor;
- Cabo Ethernet para conexão entre o servidor e o roteador.

A Figura 3.1 apresenta os principais componentes do sistema protótipo.

Figura 3.1: Sistema protótipo.




*Fonte: O AUTOR (2015).*

No Servidor foi definido o IP estático 192.168.0.8, pois é necessário que os outros dispositivos conheçam o IP do servidor, tanto para comunicação com as máquinas (Socket), quanto para comunicação com os dispositivos (web). Geralmente os servidores possuem IP estáticos para evitar estes problemas, ainda assim, futuramente podem ser criadas facilidades como utilização do serviço DNS (Domain Name System).

Para conexão do dispositivo coletor e do *tablet* ao servidor é necessário abrir o *Software* Java e clicar no botão Start, assim abrindo uma conexão servidora Socket. Através de um LED é possível identificar que o dispositivo coletor conectou na rede Wi-Fi e em seguida na conexão Socket em menos de 10 segundos depois de alimentado. O *tablet* conecta instantaneamente após aberto o aplicativo, pois já se encontra ligado e conectado a rede Wi-Fi. Com este teste pode-se afirmar o funcionamento dos itens RNF001 ao RNF004.

Com utilização da ferramenta phpMyAdmin, foram adicionadas 10 ordens de produção e 3 máquinas diferentes ao BD. Os itens RF001, RF002, RNF005 e RNF007 foram submetidos ao teste de consulta da OP no banco de dados, no qual a tela de informações da OP deve ser carregada no *tablet* em menos de 1 segundo. A velocidade de resposta foi além do esperando, carregando a tela de informações instantaneamente. Com isso concluímos que a comunicação Socket e consulta no banco de dados foram aprovados nos testes de tempo de resposta.

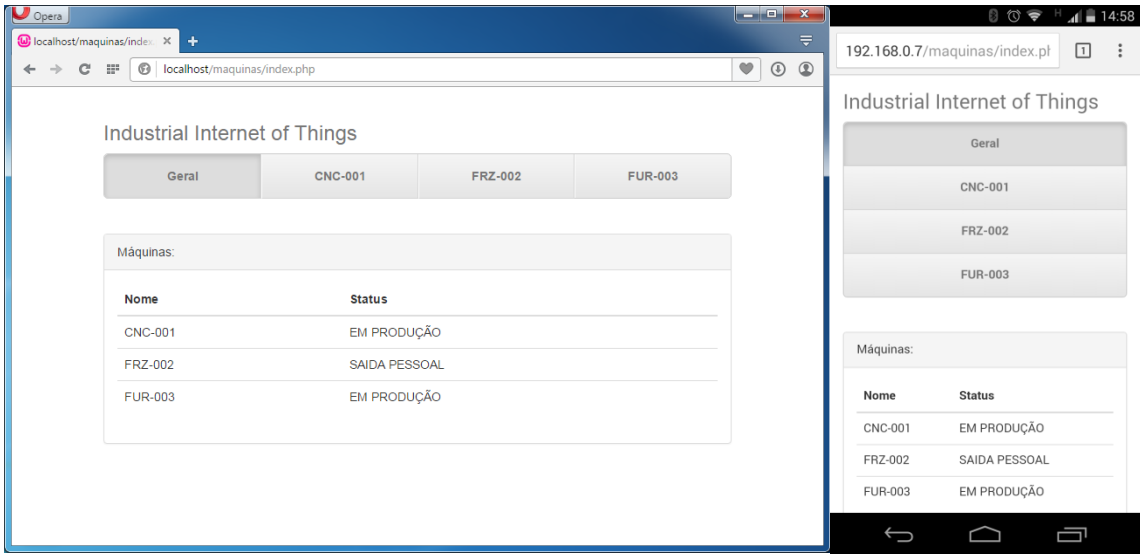
Os itens RF005 ao RF009 são referentes às interfaces e funcionalidades básicas como tela para o operador acompanhar a produção, incrementando as peças em tempo real, cronometragem e barra de progresso das peças e interface de encerramento da OP. Os itens foram testados e funcionaram corretamente. A tela de configurações foi aprovada (RF004), apesar de não possuir nenhuma configuração, seu *layout* foi desenvolvido. A tela de registro de parada das máquinas também funcionou corretamente conforme estipulado (RF003).

Com o banco de dados e servidor web online é possível testar o acesso às informações das máquinas através dos dispositivos dos gestores. Foram realizados testes em dispositivos com diferentes formatos de tela, como *desktops*, *notebooks*, *tablets* e *smartphones*. Em todos os casos a visualização foi satisfatória, e não apresentou erros de *layout*, como por exemplo, ocultar parte das tabelas. Através dos testes foram aprovados os itens RF010 e RF011, que disponibilizaram

informações de status e número da OP. O item RNF006 foi aprovado, garantindo visualização através de diferentes tamanhos de tela, conforme podem ser visualizados na Figura 3.2 os *layouts* executados em navegadores do *smartphone* Motorola e do *notebook* Sansung.

O item RNF008 foi aprovado, garantindo a atualização das informações para os gestores em um tempo inferior a 2 segundos. Porém para atualizar as informações da página é necessário atualizar a página, ou seja, é feito uma nova requisição ao servidor. Futuramente será necessário adicionar o recurso de atualização automática da página.

Figura 3.2: Representação da página HTML nos dispositivos dos gestores.




*Fonte: O AUTOR (2015).*

Com os testes realizados foi possível avaliar todos os requisitos funcionais e requisitos não funcionais descritos na Tabela 3.1. Visto que todos os requisitos foram atendidos, foi considerado que o sistema protótipo apresentou resultados satisfatórios.

# 4 CONCLUSÃO

Este trabalho propiciou uma solução tecnológica para aumentar a produtividade e eficiência de uma indústria através de um controle mais eficiente das informações de produção. Uma necessidade evidente no cenário industrial é a tomada de decisão baseadas em informação em tempo real. Aliado a esta necessidade, a comunicação sem fio tem proporcionado uma integração dos diversos dispositivos da indústria a fim de permitir esta troca de informações em tempo real.

O objetivo deste trabalho foi projetar e desenvolver um sistema baseado na arquitetura IIOT a fim de conectar dispositivos através de uma rede Wi-Fi, coletar dados das máquinas e gerenciar as informações. Este sistema é composto de três partes: um dispositivo para coleta de dados da máquina; um *tablet* para interação do operador com a máquina; um servidor para armazenar os dados e prover o acesso das informações para os *desktops* e dispositivos portáteis dos gestores. O dispositivo coletor e o *tablet* são conectados à mesma rede do servidor por um roteador Wi-Fi.

O protótipo do sistema foi avaliado conforme requisitos estabelecidos e apresentou resultados satisfatórios. Um conjunto de 11 requisitos funcionais e 8 requisitos não funcionais foram definidos com base no objetivo proposto deste trabalho. O coletor de dados foi testado utilizando um kit de desenvolvimento que simulava a ação da máquina. A IHM foi implementada em um *tablet* comercial. A visualização das informações de produção foi avaliada utilizando dispositivos móveis e um *notebook*.

Outras funcionalidades podem ser agregadas ao sistema a fim de melhorar o gerenciamento e integração dos dispositivos:

- Dispositivo coletor: comunicação protocolada com as máquinas (ModBus, CANopen, etc) e interface de cadastramento do dispositivo na rede Wi-Fi;
- Aplicativo do *tablet*: controle de usuários e configurações do sistema via *tablet*;
- Servidor: permitir integração com o sistema de gestão da empresa e desenvolver visualizações gráficas das informações coletadas.

A fim de viabilizar a utilização do sistema no setor industrial, faz-se necessário avaliar a comunicação Wi-Fi e realizar um estudo da robustez do equipamento em

relação ao ambiente industrial. Alguns dos problemas do ambiente industrial para a comunicação sem fio são as interferências geradas pelas máquinas e a grande área do chão de fábrica. Estes problemas limitam a comunicação sem fio devido à perda de pacotes no caso de interferência e a falta de conexão pela restrição do alcance da tecnologia Wi-Fi. O ambiente hostil da indústria deve ser também levado em conta na fabricação dos dispositivos utilizados neste trabalho. Assim, uma avaliação dos dispositivos deve ser realizada a fim de verificar a sua robustez quanto aos elementos do ambiente industrial.

## REFERÊNCIAS BIBLIOGRÁFICAS

ACCENTURE. **Driving Unconventional Growth through the Industrial Internet of Things**, 2015. Disponivel em: <https://www.accenture.com/mz-en/_acnmedia/Accenture/next-gen/reassembling-industry/pdf/Accenture-Driving-Unconventional-Growth-through-IIoT.pdf>. Acesso em: 2015 agosto 25.

ALFA Instrumentos. **Protocolo de Comunicação Modbus RTU/ASCII**, 2000. Disponivel em: <http://www.cerne-tec.com.br/Modbus.pdf>. Acesso em: 2015 outubro 21.

ALTERA. **Hardware Abstraction layer**, 2015. Disponivel em: <https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2sw_nii52003.pdf>. Acesso em: 2015 setembro 18.

ANDROID Dashboard, 2015. Disponivel em: <https://developer.android.com/about/dashboards/index.html?utm_source=suzunone>. Acesso em: 2015 outubro 26.

ANDROID Developer, 2015. Disponivel em: <http://developer.android.com>. Acesso em: 27 outubro 2015.

ANDROID Source, 2015. Disponivel em: <http://source.android.com/>. Acesso em: 26 outubro 2015.

ARM. **ARM Limited**, 2015. Disponivel em: <http://www.arm.com>. Acesso em: 15 setembro 2015.

BARR, M.; BROWN, C. Endianness Explained. **Barr group**, 2007. Disponivel em: <http://www.barrgroup.com/Embedded-Systems/How-To/Big-Endian-Little-Endian>. Acesso em: 25 maio 2015.

CAELUM. **PHP**, 2015. Disponivel em: <https://www.caelum.com.br/apostila-html-css-javascript/introducao-a-php/>. Acesso em: 3 novembro 2015.

CASSIOLATO, C. Wireless - ISA 100. **SMAR**, 2011. Disponivel em: <http://www.smar.com/newsletter/marketing/index131.html>. Acesso em: 25 agosto 2015.

DEITEL, P.; DEITEL, H. **Como Programar em C**. 6ª. ed. Pearson Education do Brasil, 2011.

GINATTO, A.; RAPHALOSKI, E. Protocolo WirelessHART. **Automação Industrial**, 2011. Disponivel em: <http://www.automacaoindustrial.info/o-protocolo-wirelesshart-parte-1/>. Acesso em: 25 agosto 2015.

IDC. **International Data Corporation**, 2015. Disponivel em: <http://br.idclatin.com/releases/news.aspx?id=1690>. Acesso em: 02 setembro 2015.

INFOWESTER. **Servidor Apache**, 2015. Disponivel em: <http://www.infowester.com>. Acesso em: 3 novembro 2015.

JAVAPOINT. **Java JDBC Tutorial**, 2015. Disponivel em: <http://www.javatpoint.com/java-jdbc>. Acesso em: 2 novembro 2015.

KEIL. **MDK-ARM**. Disponivel em: <http://www.keil.com/arm/mdk.asp>. Acesso em: 2015 setembro 17.

MEDEIROS, L. F. D. **Banco de dados I**. 1ª. ed. Curitiba: Intersaberes, 2013.

RIBEIRO, H. M. et al. Segurança em redes. **Revista Cientifica Eletrônica de Ciências Sociais Aplicadas da Eduvale**, Jaciara - MT, novembro 2012.

SILVA, F. A. D. Comunicação de Computadores utilizando Sockets. **FIPP**, 2008. Disponivel em: <http://www2.unoeste.br/~chico/comunicacao_socket/>. Acesso em: 12 outubro 2015.

STALLINGS, W. **Arquitetura e Organização de Computadores**. 8ª. ed. São Paulo: Pearson Pratice Hall, v. 1, 2010. Disponivel em: <http://www.ifba.edu.br/professores/antoniocarlos/index_arquivos/entradaesaida.pdf>. Acesso em: 22 maio 2015.

STANFORD. **Introduction to Computer Networking**, 2008. Disponivel em: <http://www.scs.stanford.edu/08sp-cs144/>. Acesso em: setembro 25 2015.

STM32CUBE. **ST**, 2015. Disponivel em: <http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1743/LN1897?icmp=ln1897_pron_pr_feb2015&sc=stm32cube-pr11>. Acesso em: 17 setembro 2015.

TECHNAVIO. **Industrial automation services**, 2015. Disponivel em: <http://www.technavio.com/pressrelease/global-industrial-automation-services-

market-expected-to-exceed-50-billion-by-2019-says>. Acesso em: 02 setembro 2015.

TECHNAVIO. **The Global Internet of Things Market**, 2015. Disponivel em: <http://www.technavio.com/pressrelease/the-global-internet-of-things-market-is-expected-to-grow-at-a-cagr-of-3172-percent>. Acesso em: 02 setembro 2015.

THOMÉ, G. **Tomada Eletrônica Wi-Fi**. Relatório de estágio do curso de engenharia de controle e automação. Caxias do Sul: Universidade de Caxias do Sul, 2015.

WILLIG, A. **Recent and Emerging Topics in Wireless Recent and Emerging Topics in Wireless**. 2ª. ed. Berlim, Alemanha: IEEE, v. 4, 2008.

## APÊNDICE A – Tabela de comandos do módulo Wi-Fi

Lista dos comandos “AT full Stack” do módulo Wi-Fi SPWF01Sx:

```
# AT -- Null cmd, always returns OK
# AT+CFUN =<0|1|2|3|4> -- Enable common functionalities
# AT+S. -- Switch to data mode
# AT+S.HELP -- This text
# AT&F -- Restore factory default settings
# AT&V -- Dump all settings
# AT&W -- Save current settings to flash
# AT+S.GCFG =<key> -- Get config key
# AT+S.SCFG =<key>,<value> -- Set config key
# AT+S.STS [=<sts_var>] -- Report current status/statistics
# AT+S.FSC =<fname>,<max_len> -- Create a file for httpd use
# AT+S.FSA =<fname>,<datalen><CR><data> -- Append to an existing file
# AT+S.FSD =<fname> -- Delete an existing file
# AT+S.FSL -- List existing filename(s)
# AT+S.FSP =<fname>[,<offset>,<length>] -- Print the contents of an existing file
# AT+S.GPIOC =<num>,<in|out>[,<0|R|F|B>] -- Configure specified GPIO [Optional IRQ]
# AT+S.GPIOR =<num> -- Read specified GPIO
# AT+S.GPIOW =<num>,<val> -- Write specified GPIO
# AT+S.WIFI =<0|1> -- Disable/Enable WiFi
# AT+S.ROAM -- Trigger a WiFi Roam
# AT+S.SCAN [=<a|p>,<r>] -- Perform a scan <active/passive>,<duplicated values
filter off>. Default is active,filter on
# AT+S.SSIDTXT [=<ssidtxt>] -- Set a textual SSID (not hex), otherwise prints
current SSID
# AT+S.PEERS [=peer_number[,peer_var]] -- Dump contents of the peer table
# AT+S.SOCKD =<0|port> ,<t|u> -- Disable/Enable socket server. Default is TCP
# AT+S.SOCKON =<hostname>,<port>,<t|u>[,ind] -- Open a network socket
# AT+S.SOCKQ =<id> -- Query socket for pending data
# AT+S.SOCKC =<id> -- Close socket
# AT+S.SOCKW =<id>,<len> -- Write data to socket
# AT+S.SOCKR =<id>,<len> -- Read data from socket
# AT+S.HTTPD =<0|1> -- Disable/Enable web server
# AT+S.HTTPGET =<hostname>,<path&queryopts>[,port] -- Http GET of the given path to
the specified host/port
# AT+S.HTTPPOST =<hostname>,<path&queryopts>,<formcontent>[,port] -- Http
POST of the given path to the specified host/port
# AT+S.HTTPDFSERASE -- Erase the external httpd filesystem
# AT+S.HTTPDFSUPDATE =<hostname>,<path&queryopts>[,port] -- Download a new httpd
filesystem from the specified host/port
# AT+S.FWUPDATE =<hostname>,<path&queryopts>[,port] -- Upgrade the onboard firmware
from the specified host/port
# AT+S.PING =<hostname> -- Send a ping to a specified host
```

# APÊNDICE B – Código biblioteca Wi-Fi do Coletor

## B.1 – Arquivo Wifi.h

```
#ifndef WIFI_H
	#define WIFI_H

#include "type.h"

//=============================================================================================
//-------------------- PUBLIC (Extern Variables, Constants & Defines) ---------------------
//=============================================================================================

typedef enum
{
	WIFI_OK=0,
  WIFI_ERROR,
  WIFI_TIMEOUT
}Wifi_StatusType;

typedef enum
{
	SAVE_MODE=0,
	INIT_WIFI,
	WAIT_WIFI_UP,
	WAIT_SOCKET_CONNECTION,
	SOCKET_CONNECTED,
	SOCKET_WORKING,
	NUM_OF_STATES_WIFI
}en_StateMachineWifi;

//=============================================================================================
//-------------------------- PUBLIC (Function Prototypes) -------------------------------
//=============================================================================================

void Wifi_Handler(void);
void Wifi_EngineUART(void);
void Wifi_SocketWrite(const char *buffer, uint16_t size);
void Wifi_SocketRead(void);

#endif // WIFI_H
```

## B.2 – Arquivo Wifi.c

```
//=============================================================================================
//           File: Wifi.c
//    Description: Wifi module config
//=============================================================================================
#include "type.h"
#include "monitor.h"
#include "Wifi.h"
#include "DataProtocol.h"
#include "Input.h"
#include "Output.h"
#include "Timer.h"

//------------------------------- PUBLIC (Variables) ---------------------------------

uint8_t CommunicationError=0;
uint8_t MessageWind;
uint16_t MessageData1;

//--------------------- PRIVATE (Variables, Constants & Defines) ------------------------

en_StateMachineWifi StateMachineWifi=INIT_WIFI;
```

```
typedef enum
{
	UART_MACHINE_WAITING,
	UART_MACHINE_PROTOCOL,
	UART_MACHINE_TEXTS,

}en_StateMachineUART;
en_StateMachineUART StateMachineUART=UART_MACHINE_WAITING;

#define PAD 0
#define BUFFER_SIZE 512
#define DELAY_UART 143//143ms - delay to receive the biggest frame possible

uint8_t UartRXBuffer[BUFFER_SIZE];
uint16_t UARTRXCount;

extern UART_HandleTypeDef huart1;

//Basic commands
const char RESPONSE_OK[] = "\r\nOK\r\n";
const char RESPONSE_SOCKET[] = "ID: 00\r\n\r\nOK\r\n";
const char RESPONSE_ERROR[] = "\r\nERROR: ";
const char ATENTION_MSG[] = "at\r\n";

//Module configurations commands
const char FACTORY_DEFAULT[] = "AT&F\r\n";
const char ACTIVE_STATE[] = "AT+CFUN=0\r\n";
const char RESET_MODULE[] = "AT+CFUN=1\r\n";
const char SAVE_SETTINGS[] = "AT&W\r\n";
const char ENABLE_WIFI[] = "AT+S.WIFI=1\r\n";
const char WIND_LOW[] = "AT+S.SCFG=wind_off_low,0xFFFFFFFF\r\n";
const char WIND_MED[] = "AT+S.SCFG=wind_off_medium,0xFFFFFFFF\r\n";
const char WIND_HIGH[] = "AT+S.SCFG=wind_off_high,0xFFFFFFFF\r\n";


//Network commands
const char WIFI_MODE[] = "AT+S.SCFG=wifi_mode,1\r\n";
const char SET_SSID[] = "AT+S.SSIDTXT=GVT-3FA3\r\n";//wifi name dlink router
const char PSK_PASS[] = "AT+S.SCFG=wifi_wpa_psk_text,password123\r\n";//password wifi
const char PRIV_MODE[] = "AT+S.SCFG=wifi_priv_mode,2\r\n";
const char PING[] = "AT+S.PING=192.168.1.254\r\n";
const char GET_IP[] = "AT+S.GCFG=ip_ipaddr\r\n";
const char DHCP_MODE[] = "AT+S.SCFG=ip_use_dhcp,1\r\n";
const char SET_HOSTNAME[] = "AT+S.SCFG=ip_hostname,collector\r\n";

const char SOCKET_ENABLE[] = "AT+S.SOCKON=192.168.0.8,8080,t\r\n";
const char SOCKET_WRITE[] = "AT+S.SOCKW=00,";
const char SOCKET_READ[] = "AT+S.SOCKR=00,";
const char SOCKET_QUERYDATA[] = "AT+S.SOCKQ=00\r\n";
const char CHANGE_MODE[] = "AT+S.\r\n";

//GPIO commands
const char GPIO_CONFIG[] = "AT+s.gpioc=14,out\r\n";
const char GPIO_ON[] = "AT+s.gpiow=14,1\r\n";
const char GPIO_OFF[] = "AT+s.gpiow=14,0\r\n";

//Module Responses
const char RES_MODULE_READY[] = "+WIND:0";
const char RES_WIFI_UP[] = "+WIND:24:WiFi Up:";
const char RES_INCOMING_SOCKET[] = "+WIND:60:Now in Data Mode";

//------------------------------ PRIVATE (Function Prototypes) ----------------------------

void Wifi_ClearBuffer(void)
{
```

```
	memset(UartRXBuffer,PAD,BUFFER_SIZE);
}

void Wifi_Transmit(const char* buffer, uint16_t size)
{
	HAL_UART_Transmit(&huart1, (uint8_t *)buffer , size, 1000);
}

Wifi_StatusType Wifi_WaitResponseCommand(const char *ptr)
{
	UARTRXCount=0;
	HAL_UART_Receive_IT(&huart1, &UartRXBuffer[0], BUFFER_SIZE, &UARTRXCount);
	Counter_ms=0;
	do
	{
		//searching a string RESPONSE_OK inside the array
		if(strstr((const char*) UartRXBuffer, ptr) != NULL)
		{
			xprintf(" OK");
			Wifi_ClearBuffer();
			return WIFI_OK;
		}
		else if(strstr((const char*) UartRXBuffer, RESPONSE_ERROR) != NULL)
		{
			xprintf(" ERROR");
			Wifi_ClearBuffer();
			return WIFI_ERROR;
		}
	}
	while(Counter_ms < DELAY_UART);

	Wifi_ClearBuffer();
	xprintf(" TIMEOUT");
	return WIFI_TIMEOUT;//communication problem
}


Wifi_StatusType Wifi_SendCommand(const char* buffer, const char* resPtr)
{
	Wifi_StatusType res;

	xprintf("\nSend Command: %s",buffer);
	Wifi_Transmit(buffer, (uint16_t)strlen(buffer));
	res = Wifi_WaitResponseCommand(resPtr);
	if(res == WIFI_TIMEOUT)
	{
		Wifi_Transmit(buffer, (uint16_t)strlen(buffer));
		res = Wifi_WaitResponseCommand(resPtr);
		if(res == WIFI_TIMEOUT)
		{
			CommunicationError = 1;
		}
	}
	return res;
}

//=========================================================================================
//-------------------------------- Public Functions ---------------------------------------
//=========================================================================================

void Wifi_Handler(void)
{
	switch(StateMachineWifi)
	{
		case INIT_WIFI:
			if(Wifi_SendCommand(ATENTION_MSG, RESPONSE_OK) == WIFI_OK)
```

```
                        {
                                UARTRXCount=0;
                                HAL_UART_Receive_IT(&huart1, UartRXBuffer, BUFFER_SIZE,
&UARTRXCount);//able uart interrupt
                                Wifi_SendCommand(GPIO_CONFIG, RESPONSE_OK);
                                StateMachineWifi = WAIT_WIFI_UP;
                                xprintf("\nWifi Comunication OK... Initialiazed");
                        }
                        break;

                case WAIT_WIFI_UP:
                        if(MessageWind == 24)//Wifi Up
                        {
                                xprintf("\nWiFi Up");
                                StateMachineWifi++;
                        }
                        break;

                case WAIT_SOCKET_CONNECTION:
                        if(Wifi_SendCommand(SOCKET_ENABLE, RESPONSE_SOCKET) == WIFI_OK)
                        {
                                StateMachineWifi++;
                                xprintf("\nSocket working...");
                        }
                        break;

                case SOCKET_CONNECTED:
                        StateMachineWifi++;
                        DataProtocol_SendID();
                        Output_Set(LED4, TURN_ON_OUT);
                        break;

                case SOCKET_WORKING:
                        Wifi_SocketRead();
                        break;

                case SAVE_MODE:
                default:
                        if(Wifi_SendCommand(ATENTION_MSG, RESPONSE_OK) == WIFI_OK)
                        {
                                xprintf("\nCommunication restored...");
                                StateMachineWifi = INIT_WIFI;
                        }
                        break;
        }

        //Key
        if(Input_Get(B1) == HOLD_EVENT_IN)
        {
                Input_Set(B1,NONE_IN);
                //Factory Default
                if(Wifi_SendCommand(FACTORY_DEFAULT, RESPONSE_OK) == WIFI_OK)
                {
                        xprintf("\nFactory Default Parameters");

                        Wifi_SendCommand(SET_SSID, RESPONSE_OK);
                        Wifi_SendCommand(PSK_PASS, RESPONSE_OK);
                        Wifi_SendCommand(PRIV_MODE, RESPONSE_OK);
                        Wifi_SendCommand(DHCP_MODE, RESPONSE_OK);
                        Wifi_SendCommand(SET_HOSTNAME, RESPONSE_OK);
                        Wifi_SendCommand(WIFI_MODE, RESPONSE_OK);

                        Wifi_SendCommand(SAVE_SETTINGS, RESPONSE_OK);
                        Wifi_SendCommand(RESET_MODULE, RES_MODULE_READY);
                        xprintf("\nInitializing Module");
```

```
                        StateMachineWifi=INIT_WIFI;
                }
        }
        if(Input_Get(B1) == RELEASE_EVENT_IN)
        {
                //Count piece
                if(StartProduction)
                {
                        CounterPieces++;
                        DataProtocol_SendInfoPieces();
                }
                Input_Set(B1,NONE_IN);
        }
}


void Wifi_EngineUART(void)
{
        static uint16_t countByte;
        uint16_t index;

        if (UARTRXCount>0)
        {
                for(index=0; index<UARTRXCount; index++)
                {
                        xprintf("%c",UartRXBuffer[index]);

                        switch(StateMachineUART)
                        {
                                case UART_MACHINE_WAITING:
                                        if(UartRXBuffer[index] == '+')
                                        {
                                                countByte=0;
                                                StateMachineUART = UART_MACHINE_TEXTS;
                                        }
                                        break;

                                case UART_MACHINE_TEXTS:
                                        countByte++;
                                        if(countByte == 6)
                                        {
                                                MessageWind = (UartRXBuffer[index]-0x30)*10;
                                        }
                                        else if(countByte == 7)
                                        {
                                                MessageWind += (UartRXBuffer[index]-0x30);
                                                        StateMachineUART = UART_MACHINE_WAITING;
                                        }
                                        break;

                                case UART_MACHINE_PROTOCOL:
                                        DataProtocol_Trate(UartRXBuffer[index]);
                                        break;

                                default:
                                        break;
                        }
                }
                UARTRXCount = 0;
                HAL_UART_Receive_IT(&huart1, UartRXBuffer, BUFFER_SIZE, &UARTRXCount);
        }
}

void Wifi_SocketWrite(const char *buffer, uint16_t size)
{
        char str1[25];
```

```
	char str2[3];

	if(size>0)
	{
		strcpy (str1,SOCKET_WRITE);
		itoa(size,str2);
		strncat (str1, str2, 14);
		strncat (str1, "\r", 14+findn(size));

		Wifi_Transmit(str1, (uint16_t)strlen(str1));
		Wifi_Transmit(buffer, size);
	}
}

Wifi_StatusType Wifi_ReadQueryData(void)
{
	char * pch;
	UARTRXCount=0;
	HAL_UART_Receive_IT(&huart1, &UartRXBuffer[0], BUFFER_SIZE, &UARTRXCount);
	Counter_ms=0;
	do
	{
		//searching a string inside the array
		pch = strstr((const char*) UartRXBuffer, "DATALEN: ");
		if(pch != NULL)
		{
			HAL_Delay(5);

			if((pch[9]) != '\r')
			{
				MessageData1 = pch[9]-0x30;
				if((pch[10]) != '\r')
				{
					MessageData1 *=10;
					MessageData1 += pch[10]-0x30;
					if((pch[11]) != '\r')
					{
						MessageData1 *=10;
						MessageData1 += pch[11]-0x30;
					}
				}
			}
			if(MessageData1 == 0)
				Wifi_ClearBuffer();
			return WIFI_OK;
		}
	}
	while(Counter_ms < DELAY_UART);


	Wifi_ClearBuffer();
	xprintf(" TIMEOUT");
	return WIFI_TIMEOUT;//communication problem
}

void Wifi_SocketRead(void)
{
	char str1[25];
	char str2[3];

	Wifi_Transmit(SOCKET_QUERYDATA, (uint16_t)strlen(SOCKET_QUERYDATA));
	if(Wifi_ReadQueryData() == WIFI_OK)
	{
		if(MessageData1 > 0)
		{
			strcpy (str1,SOCKET_READ);
```

```
                        itoa(MessageData1,str2);
                        strncat (str1, str2, 14);
                        strncat (str1, "\r", 14+findn(MessageData1));

                        Wifi_Transmit(str1, (uint16_t)strlen(str1));

                        StateMachineUART = UART_MACHINE_PROTOCOL;
                }
        }
}
```

# APÊNDICE C – Código do Aplicativo para Android

## C.1 – Arquivo XML main

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:focusable="false">

        <Button
            style="?android:attr/borderlessButtonStyle"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/ProdButton"
            android:text="Iniciar Produção"
            android:textAlignment="center"
            android:drawableTop="@drawable/pro_icon"
            android:gravity="center"
            android:layout_marginRight="100sp"
            android:layout_gravity="center" />

        <Button
            style="?android:attr/borderlessButtonStyle"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/MainButton"
            android:text="Registar Parada"
            android:textAlignment="center"
            android:drawableTop="@drawable/main_icon"
            android:gravity="center"
            android:layout_marginLeft="100sp"
            android:layout_gravity="center" />
    </FrameLayout>

    <Button
        style="?android:attr/borderlessButtonStyle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/SettingButton"
        android:text="Configuraçõe"
        android:textAlignment="center"
        android:drawableTop="@drawable/set_icon"
        android:layout_alignParentLeft="true"
        android:layout_marginLeft="20sp"
        android:layout_marginTop="20sp" />

</RelativeLayout>
```

## C.2 – Arquivo XML main

```
package com.example.guithome.iiot;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.widget.Button;

public class Collector extends Activity {

    private InitProduction dInitProduction = null;
    Button settingButton;
    Button initProdButton;
```

```
    Button mainButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(com.example.guithome.iiot.R.layout.main);

        settingButton = ((Button) findViewById(R.id.SettingButton));
        logoutButton = ((Button) findViewById(R.id.LogoutButton));
        initProdButton = ((Button) findViewById(R.id.ProdButton));
        mainButton = ((Button) findViewById(R.id.MainButton));

        settingButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent myIntent = new Intent(Collector.this, ItemListActivity.class);
                startActivity(myIntent);
            }
        });

        initProdButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(Collector.this, InitProduction.class);
                startActivity(intent);
            }
        });

        mainButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(Collector.this, RegStopped.class);
                startActivityForResult(intent, 1);
            }
        });

        ((Protocol) this.getApplication()).SetStatusListener(new Protocol.StatusListener()
{
            @Override
            public void StartActivityProductionOrder(classOcObject obj) {
            }

            @Override
            public void ChangeNumberPieces(int num) {
                Intent intent = new Intent(Collector.this, ProductionOrder.class);
            }
        });
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        ((Protocol) this.getApplication()).Stop();
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        ((Protocol) this.getApplication()).Resume();
    }
}
```

# APÊNDICE D – Código do Servidor Java

## D.1 – Arquivo UserManager.java

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;

public class UserManager extends Thread {

    // contains information about the current user
    private User user;
    // the socket that links the user(client) to this server
    private Socket socket;
    private PrintWriter bufferSender;
    // flag used to stop the read operation
    private boolean running;
    // used to notify certain user actions like receiving a message or disconnect
    private UserManagerDelegate managerDelegate;

    public UserManager(Socket socket, UserManagerDelegate managerDelegate) {
        this.user = new User();
        this.socket = socket;
        this.managerDelegate = managerDelegate;
        running = true;
    }

    public User getUser() {
        return user;
    }

    public Socket getSocket() {
        return socket;
    }

    @Override
    public void run() {
        super.run();

        System.out.println("S: Receiving...");

        try {

            //sends the message to the client
            bufferSender = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream())), true);

            //read the message received from client
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

            //in this while we wait to receive messages from client (it's an infinite loop)
            //this while it's like a listener for messages
            while (running) {

                int message = 0xFFFF;
                try {
                    message = in.read();
                } catch (IOException e) {
                    System.out.println("Error reading message: " + e.getMessage());
                }
```

```
                if (message != 0xFFFF && managerDelegate != null) {
                      user.setintMessage(message);
                    // notify message received action
                    managerDelegate.messageReceived(user);
                }
            }

        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    public void close() {
        running = false;

        if (bufferSender != null) {
            bufferSender.flush();
            bufferSender.close();
            bufferSender = null;
        }

        try {
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void sendMessage(String message) {
        if (bufferSender != null && !bufferSender.checkError()) {
            bufferSender.println(message);
            bufferSender.flush();
        }
    }

    public interface UserManagerDelegate {

        /**
         * Called when the manager receives a new message from the client
         */
        public void messageReceived(User fromUser);
    }
}
```

## D.2 – Arquivo TcpServer.java

```
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;

public class TcpServer extends Thread implements UserManager.UserManagerDelegate {

    public static final int SERVERPORT = 8080;
    // while this is true the server will run
    private boolean running = false;
    // callback used to notify new messages received
    private ServerSocket serverSocket;
    private ArrayList<UserManager> connectedUsers;
    private MsgReceivedInterface listener;

    public TcpServer() {
        connectedUsers = new ArrayList<UserManager>();
    }
```

```
    public void SetStatusListener(MsgReceivedInterface listener){
        this.listener = listener;
    }

    public void close() {
        running = false;

        try {
            serverSocket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        System.out.println("S: Done.");
        serverSocket = null;
    }

    public void sendMessage(String str, int id) {

        if (connectedUsers != null) {
            for (UserManager userManager : connectedUsers) {
                if (userManager.getUser().getUserID() == id) {
                    userManager.sendMessage(str);
                }
            }
        }
    }

    private void runServer() {
        running = true;

        try {
            //create a server socket. A server socket waits for requests to come in over
the network.
            serverSocket = new ServerSocket(SERVERPORT);

            while (running) {
                // Create a loop and get all the incoming connections and create users

                System.out.println("S: Waiting for a client ...");

                // Listens for a connection to be made to this socket and accepts it.
                Socket client = serverSocket.accept();// Create client socket...

                UserManager userManager = new UserManager(client, this);
                // add the new user to the stack of users
                connectedUsers.add(userManager);

                // start reading messages from the client
                userManager.start();
                System.out.println("S: New client connected ...");
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void run() {
        super.run();
        runServer();
    }

    @Override
```

```
    public void messageReceived(User fromUser) {

        //received message
        if (listener != null) {
                listener.messageReceived(fromUser);
         }
    }

    public interface MsgReceivedInterface {

         /**
          * Called when the manager receives a new message from the client
          */
         public void messageReceived(User fromUser);
    }
}
```

## D.3 – Arquivo DataBase.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public final class DataBase {
  private Connection connect = null;
  private Statement statement = null;
  private PreparedStatement preparedStatement = null;
  private ResultSet resultSet = null;

  public void connectDB() throws Exception {
  // This will load the MySQL driver, each DB has its own driver
  Class.forName("com.mysql.jdbc.Driver");
  // Setup the connection with the DB
  connect = DriverManager
      .getConnection("jdbc:mysql://localhost/database?"
          + "user=root");
  }

  public boolean searchOP(int number) throws SQLException{
         try {
                  connectDB();
                  // Statements allow to issue SQL queries to the database
                  statement = connect.createStatement();
                  // Result set get the result of the SQL query
                  preparedStatement = connect
                            .prepareStatement("select name from database.op where number =
?");
                preparedStatement.setInt(1, number);

                resultSet = preparedStatement.executeQuery();

                while (resultSet.next()) {
                        System.out.println("Encontrou");
                        return true;
                }
                System.out.println("Não Encontrou");
                close();
                return false;
         } catch (Exception e) {
                try {
                     throw e;
              } catch (Exception e1) {
```

```
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
            }
        return false;
}

  public OrderProducion getOP(int number) throws SQLException{
          try {
                  OrderProducion prod = new OrderProducion();
                  // Statements allow to issue SQL queries to the database
              statement = connect.createStatement();
              // Result set get the result of the SQL query
              preparedStatement = connect
                              .prepareStatement("select status, number, code, name, quantity,
start, delivery"
                              + " from database.op where number = ?");
                   preparedStatement.setInt(1, number);

                   resultSet = preparedStatement.executeQuery();

              while (resultSet.next()) {
                prod.setStatus(resultSet.getInt("status"));
                prod.setNumber(resultSet.getString("number"));
                 prod.setProductCode(resultSet.getString("code"));
                 prod.setProductName(resultSet.getString("name"));
                 prod.setQuantity(resultSet.getInt("quantity"));
                 prod.setStart(resultSet.getDate("start"));
                 prod.setDelivery(resultSet.getDate("delivery"));
              }
              return prod;

          } catch (Exception e) {
                  throw e;
          } finally {
                  close();
          }
  }

  public void writeOP(OrderProducion prod) throws SQLException{
          try {

                  preparedStatement = connect
                   .prepareStatement("insert into  database.op values
(default,?,?,?,?,?,?,?)");

              preparedStatement.setInt(1, prod.getStatus());
              preparedStatement.setString(2, prod.getNumber());
              preparedStatement.setString(3, prod.getProductCode());
              preparedStatement.setString(4, prod.getProductName());
              preparedStatement.setInt(5, prod.getQuantity());
              preparedStatement.setDate(6, new java.sql.Date(prod.getStartYear(),
prod.getStartMonth(), prod.getStartDay()));
                          //Calendar.getInstance().getTimeInMillis()));
              preparedStatement.setDate(6, prod.getStart());
              preparedStatement.setDate(7, prod.getDelivery());
              preparedStatement.executeUpdate();
          } catch (Exception e) {
                  throw e;
          } finally {
                  close();
          }
}

  private void close() {
        try {
```

```
      if (resultSet != null) {
        resultSet.close();
      }
      if (statement != null) {
        statement.close();
      }
      if (connect != null) {
        connect.close();
      }
    } catch (Exception e) {}
  }
}
```

# APÊNDICE E – Código do Servidor PHP

## E.1 – Arquivo index.php

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
  <meta http-equiv="Content-Type" content="text/html"; charset=iso-8859-1">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap core CSS -->
    <link href="assets/dist/css/bootstrap.min.css" rel="stylesheet">

    <!-- Custom styles for this template -->
    <link href="assets/docs/examples/justified-nav/justified-nav.css" rel="stylesheet">

    <!-- Just for debugging purposes. Don't actually copy these 2 lines! -->
    <script src="assets/js/ie-emulation-modes-warning.js"></script>

  </head>
  <body>

  <?php

  //conectando com o localhost - mysql
  $conexao = mysqli_connect("localhost","root");
  if (!$conexao)
         die ("Erro de conexão com localhost, o seguinte erro ocorreu -> ".mysql_error());
  //conectando com a tabela do banco de dados
  $banco = mysqli_select_db($conexao,"database");
  if (!$banco)
         die ("Erro de conexão com banco de dados, o seguinte erro ocorreu ->
".mysql_error());

  if(isset($_GET['id'])) {
    $idUser = $_GET["id"];

    $query = "SELECT status,code FROM op WHERE id = " . + $idUser;
    $result = mysqli_query($conexao,$query); // run the query and get the result object.
    if (!$result) { // check for errors.
      echo 'Could not run query: ' . mysql_error();
      exit;
    }
    $row = mysqli_fetch_row($result); // get the single row.
    mysqli_free_result($result);
  } else {
    $idUser = 0;
    $query = "SELECT * FROM op";
    $result = mysqli_query($conexao,$query); // run the query and get the result object.
    if (!$result) { // check for errors.
      echo 'Could not run query: ' . mysql_error();
      exit;
      }
   }
?>

  <div class="container">

    <!-- The justified navigation menu is meant for single line per list item.
         Multiple lines will require custom code not provided by Bootstrap. -->
    <div class="masthead">
      <h3 class="text-muted">Industrial Internet of Things</h3>
      <nav>
        <ul class="nav nav-justified">
```

```
          <li_<?php if ($idUser== 0) echo 'class="active"'?>><a
href="index.php">Geral</a></li>
          <li_<?php if ($idUser==1) echo 'class="active"'?>><a
href="index.php?id=1">Máquina 1</a></li>
          <li_<?php if ($idUser==2) echo 'class="active"'?>><a
href="index.php?id=2">Máquina 2</a></li>
          <li_<?php if ($idUser==3) echo 'class="active"'?>><a
href="index.php?id=3">Máquina 3</a></li>
        </ul>
      </nav>
    </div>

<?php
  if ($idUser== 0) {
?>

    </br></br>
    <div class="panel panel-default">
    <!-- Default panel contents -->
    <div class="panel-heading">Máquinas</div>
    <div class="panel-body">
      <p> Essas são as máquinas </p>
    </div>
    <table class="table">
    <thead>
      <tr>
          <th data-field="id">Item ID</th>
          <th data-field="status">Status</th>
          <th>Ordem de Produção</th>
          <th>Código</th>
          <th>Nome</th>
          <th>Quantidade</th>
          <th>Data de Emissão</th>
          <th>Data de Entrega</th>
      </tr>
  </thead>

<?php
    while ($row = mysqli_fetch_row($result)) {
     if (!empty($row)){
?>
        <tr>
        <td> <?php echo $row[0] ?> </td>
        <td> <?php echo $row[1] ?> </td>
        <td> <?php echo $row[2] ?> </td>
        <td> <?php echo $row[3] ?> </td>
        <td> <?php echo $row[4] ?> </td>
        <td> <?php echo $row[5] ?> </td>
        <td> <?php echo $row[6] ?> </td>
        <td> <?php echo $row[7] ?> </td>
        </tr>
<?php
        }
    }
?>
  </table>
</div>

<?php
  }
?>
    <!-- IE10 viewport hack for Surface/desktop Windows 8 bug -->
    <script src="assets/js/ie10-viewport-bug-workaround.js"></script>
  </body>
</html>
```