Felipe Rodrigues Pereira Fonseca

Desenvolvimento de uma Plataforma Embarcada para Monitoramento e Operação de um Trocador de Calor

Belo Horizonte

Novembro de 2017

Felipe Rodrigues Pereira Fonseca

Desenvolvimento de uma Plataforma Embarcada para Monitoramento e Operação de um Trocador de Calor

Monografia submetida à banca examinadora designada pelo Colegiado Didático do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Minas Gerais,como parte dos requisitos para aprovação na disciplina Projeto Final de Curso II.

Universidade Federal de Minas Gerais

Escola de Engenharia

Curso de Graduação em Engenharia de Controle e Automação

Orientador: Lúcio Fábio Dias Passos, DEQ/UFMG

Belo Horizonte

Novembro de 2017

Felipe Rodrigues Pereira Fonseca

Desenvolvimento de uma Plataforma Embarcada para Monitoramento e Operação de um Trocador de Calor/ Felipe Rodrigues Pereira Fonseca. – Belo Horizonte, Novembro de 2017-

44 p. : il. (algumas color.) ; 30 cm.

Orientador: Lúcio Fábio Dias Passos, DEQ/UFMG

Monografia de Projeto Final de Curso – Universidade Federal de Minas Gerais
Escola de Engenharia
Curso de Graduação em Engenharia de Controle e Automação, Novembro de 2017.

1. Palavra-chave1. 2. Palavra-chave2. I. Orientador. II. Universidade xxx. III. Faculdade de xxx. IV. Título

CDU 02:141:005.7

Felipe Rodrigues Pereira Fonseca

Desenvolvimento de uma Plataforma Embarcada para Monitoramento e Operação de um Trocador de Calor

Monografia submetida à banca examinadora designada pelo Colegiado Didático do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Minas Gerais,como parte dos requisitos para aprovação na disciplina Projeto Final de Curso II.

Trabalho aprovado. Belo Horizonte, 24 de novembro de 2012:

Lúcio Fábio Dias Passos, DEQ/UFMG
Orientador

Professor
Convidado 1

Professor
Convidado 2

Belo Horizonte
Novembro de 2017

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

Agradecimentos

Os agradecimentos principais são direcionados à Gerald Weber, Miguel Frasson, Leslie H. Watter, Bruno Parente Lima, Flávio de Vasconcellos Corrêa, Otavio Real Salvador, Renato Machnievscz[1] e todos aqueles que contribuíram para que a produção de trabalhos acadêmicos conforme as normas ABNT com LaTeX fosse possível.

Agradecimentos especiais são direcionados ao Centro de Pesquisa em Arquitetura da Informação[2] da Universidade de Brasília (CPAI), ao grupo de usuários *latex-br*[3] e aos novos voluntários do grupo *abnTeX2*[4] que contribuíram e que ainda contribuirão para a evolução do abnTeX2.

[1] Os nomes dos integrantes do primeiro projeto abnTeX foram extraídos de http://codigolivre.org.br/projects/abntex/

[2] http://www.cpai.unb.br/

[3] http://groups.google.com/group/latex-br

[4] http://groups.google.com/group/abntex2 e http://abntex2.googlecode.com/

"Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12, 2)

Resumo

Resumo

Palavras-chaves: latex. abntex. editoração de texto.

Abstract

This is the english abstract.

Key-words: latex. abntex. text editoration.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

Fig.	Area of the i^{th} component
456	Isto é um número
123	Isto é outro número
lauro cesar	este é o meu nome

Lista de símbolos

Γ	Letra grega Gama
Λ	Lambda
ζ	Letra grega minúscula zeta
$\in$	Pertence

Lista de Códigos

Sumário

1 Introdução

Este capítulo apresenta a motivação e justificativa para a realização deste projeto e o local de desenvolvimento, bem como descreve brevemente a estrutura da monografia

1.1 Motivação e Justificativa

Atualmente observa-se uma crescente influência dos sistemas embarcados[1] em diferentes segmentos. Estes sistemas, que antes eram utilizados apenas em equipamentos complexos, estão presentes também em muitos dispositivos comuns no cotidiano das pessoas. É possível verificar a sua utilização em câmeras, eletrodomésticos, brinquedos e também no setor automotivo, em máquinas industriais, dispositivos da indústria de processos entre outros (WANKE; HUMPHREY, 2017; PEREIRA; CARVALHO et al., 2011). A utilização de sistema embarcado é incentivada à medida que a evolução tecnológica proporciona maior poder de processamento em dispositivos de tamanho cada vez menor (GREENFIELD, 2013).

Um trocador de calor pode ser definido como um dispositivo em que ocorre uma transferência de calor entre duas substâncias que estejam em temperaturas distintas. Geralmente, as substâncias envolvidas são fluidos. Os trocadores de calor podem ser classificados com relação a diversos critérios, como por exemplo o modo de troca de calor, tipo de construção, entre outros. (KREITH; MANGLIK; BOHN, 2011)

Indústrias dos mais variados setores utilizam trocadores de calores para diversas funcionalidades, tais como: recuperar energia térmica gerada em algum processo, com o intuito de reduzir o consumo de energia da planta e transportar produtos em temperaturas predeterminadas.

Portanto, é de suma importância a utilização de um sistema de controle eficiente em plantas que contém trocadores de calor. Plantas como essa devem ser capazes de atender referências de temperatura bem como rejeitar de possíveis pertubações que podem ocorrer durante o funcionamento do processo. (NOVAZZI, 2007). Além disso, soluções que permitem o monitoramento e operação remota das plantas são relevantes para o processo, uma vez que trocadores de calor podem estar instalados em ambientes de difícil acesso ou operando em altas temperaturas. A operação remota de plantas industriais traz outros benefícios como (KIRUBASHANKAR; K; J, 2009):

- Interface amigável de operação, que contribui para uma atuação mais rápida e assertiva;

[1] http://files.comunidades.net/mutcom/ARTIGO_SIST_EMB.pdf

- Melhor acessibilidade aos dados coletados, o que permite a equipe de engenharia e gestão analisar e propor melhorias na operação.

Diferentes arquiteturas de sistema de supervisão (monitoramento e operação) e controle podem ser aplicadas em plantas industriais. Encontram-se plantas utilizando a tradicional topologia formada por PLC e um computador (PC) executando alguma ferramenta SCADA [2], bem como é possível verificar a utilização de sistemas embarcados. A escolha entre uma ou outra arquitetura depende de uma série de fatores, como: funcionalidades disponibilizadas por cada ferramenta, custo, domínio do desenvolvedor para utilizar determinados softwares, entre outros.

Um sistema embarcado pode ser definido como um sistema projetado para realizar tarefas específicas e geralmente fazem parte de uma aplicação maior, operando em tempo real e sem intervenção do usuário (BASKIYAR; MEGHANNATHAN, 2005). Devido ao avanço da tecnologia e o aumento do poder de processamento dos hardwares, os sistemas embarcados podem realizar cada vez mais tarefas, ou seja, disponibilizar mais funcionalidades aos usuários (PEREIRA; CARVALHO et al., 2011). Este fenômeno também ocorre para os softwares SCADA, que contam com maior capacidade de processamento e possuem mais funcionalidades (GREENFIELD, 2017). Porém, quanto maior é a variabilidade de opções de projeto disponível, maior é a dificuldade de se alcançar uma padronização no desenvolvimento e, principalmente, uma padronização da comunicação entre os componentes, pois muitos sistemas ainda se comunicam em protocolos proprietários.

A demanda crescente pela integração de sistemas (por exemplo integração entre sistemas SCADAs, ERPs e MES), incentivou o desenvolvimento de funcionalidades para prover essa comunicação. (POLÔNIA, 2011; VALENZUELA et al., 2013). As potencialidades da World Wide Web, rede que se expande rapidamente desde sua origem, têm sido estudadas como ferramentas para desenvolvimento de sistemas flexíveis, modulares e interoperáveis (POLÔNIA, 2011 apud FIELDING, 2000). A Web, desde a sua criação se baseia em tecnologias abertas e padronizadas que podem auxilar o processo de desenvolvimento desses sistemas.

Este trabalho apresenta o desenvolvimento de um sistema baseado em tecnologia Web sobre um dispositivo embarcado de baixo custo para monitoramento e operação de um trocador de calor, que está localizado em um dos laboratórios do departamento da engenharia química da UFMG. Para isso, foram utilizados protocolos de comunicação e ferramentas de código aberto, para possibilitar futura integração de outros sistemas e/ou funcionalidades.

O tema é relevante pois, a implementação proposta permite aos alunos que realizem as práticas no laboratório forma mais adequada, de forma que os esforços estejam

[2] https://www.wonderware.com/hmi-scada/what-is-scada/

concentrados na análise dos experimentos e identificação dos fenômenos, e não na coleta de dados e regulação da planta nos pontos de operação. Essa implementação também permite aos alunos de Engenharia de Controle e Automação estudar modelagem de sistemas e projeto de controladores. Por fim, a solução proposta também proporciona os alunos o contato com tecnologias emergentes e que podem ser utilizadas largamente nas indústrias.

1.2 Objetivos

De acordo com as informações expostas acima, este Projeto Final de Curso (PFC) possui o seguinte objetivo geral:

- Propor e testar uma solução baseada em sistemas embarcados para monitoramento e operação remota de um trocador de calor.

Além do objetivo geral, o projeto possui também os seguintes objetivos específicos:

- Comparar a solução embarcada com um software de prateleira tradicional;
- Proporcionar uma base de conhecimento para implementações de sistemas semelhantes para outras plantas do laboratório;
- Possibilitar a inserção de mais funcionalidades ao sistema, como por exemplo um algoritmo de autovalidação. da planta;

1.3 Local de Realização

O projeto foi desenvolvido no Laboratório de Operações e Processos Industriais, um dos laboratórios que pertencem ao Departamento de Engenharia Química (DEQ) da UFMG. Estes laboratórios são utilizados para fins didáticos e de pequisa.

1.4 Estrutura da Monografia

O trabalho está dividido em 6 capítulos. O presente capítulo apresentou a motivação e os objetivos do trabalho. O capítulo 2 ....

2 Revisão Bibliográfica

Teste, teste

2.1 Laboratório de Operações Unitárias - Contextualização

O Laboratório de Operações Unitárias (LOU-DQ) é um dos diversos laboratórios para fins de ensino e pesquisa pertencentes ao Departamento de Engenharia Química da UFMG (DEQ). Ainda existem outros laboratórios vinculados ao departamento cuja finalidade é prestar serviços às comunidades interna e externa à UFMG [1].

O LOU possui quatro plantas didáticas: um sistema de trocador de calor, uma planta que implementa um processo de sedimentação, uma planta piloto de um hidrociclone e uma planta que implementa um processo de secagem.

O trocador de calor existente no laboratório, cuja imagem pode ser vista na Figura 2.1, é do tipo tubular. Este sistema, que é fabricado e fornecido pela Universidade de Santa Cecília[2], foi adquirido pela UFMG em 2002[3]. Originalmente, a planta possui um botão para ligar e desligar bomba e aquecedor simultaneamente, bem como conta com 4 sensores de temperatura, cujos valores são exibidos e display LCDs. A medição de vazão de água quente era inferida através da utilização de uma Calha Parshall[4], e a medição de água fria, através da indicação de um rotâmetro[5]

Figura 2.1 – Trocador de Calor presente no LOU.

[1] http://www.deq.ufmg.br/departamento/infraestrutura
[2] http://cursos.unisanta.br/quimica/laborato/index.html
[3] Item 35 do catálogo disponível em: http://cursos.unisanta.br/quimica/laborato/LOU-2016.pdf
[4] Funcionamento da Calha Parshall: http://www.dec.ufcg.edu.br/saneamento/PARSHALL.html
[5] http://www.sensorsmag.com/components/basics-rotameters

Em 2016, um projeto de modernização desta planta foi iniciado por (PEREIRA; ROCHA et al., 2016). O projeto consistiu na implementação de um sistema embarcado para operação local e controle da planta. Para alcançar o objetivo, foram feitos os seguintes passos:

- Instalação de quatro novos sensores de temperatura, dois novos sensores de vazão, tornando possível a medição digital das vazões; 2 relés, para o controle do acionamento dos atuadores e um inversor de frequência para controlar a rotação da bomba;
- Instalação de um Arduino Due [6] para fazer a interface com sensores e atuadores, processar os algoritmos de controle, e exibir os dados em um display LCD. O programa criado no Arduino possibilita a operação em modo automático e manual;
- Construção e instalação de um painel para operar e visualizar os dados da planta. Através desse painel é possível selecionar o modo de funcionamento da planta (manual ou automático). Em modo manual, é possível ligar e desligar bomba e aquecedor, bem como controlar a velocidade da bomba e potência do aquecedor. O display LCD permite exibir várias informações como por exemplo valores das temperaturas e vazões, bem como a sintonia dos controladores. Essas informações não conseguem ser expostas ao mesmo tempo para o usuário, portanto um sistema de navegação foi implementado no Arduino. O esboço do painel elaborado é mostrado na Figura 2.2.

Figura 2.2 – Painel instalado na planta de trocador de calor.
Adaptado de (PEREIRA; ROCHA et al., 2016)

A Arquitetura do sistema finalizado é exibida na Figura 2.3. Destaca-se o fato que não foi possível integrar o controle da potência do aquecedor no Arduino, sendo enviado

[6] https://store.arduino.cc/usa/arduino-due

diretamente do painel. Apesar de previsto no projeto, nenhum algoritmo para controle em malha fechada foi programado.

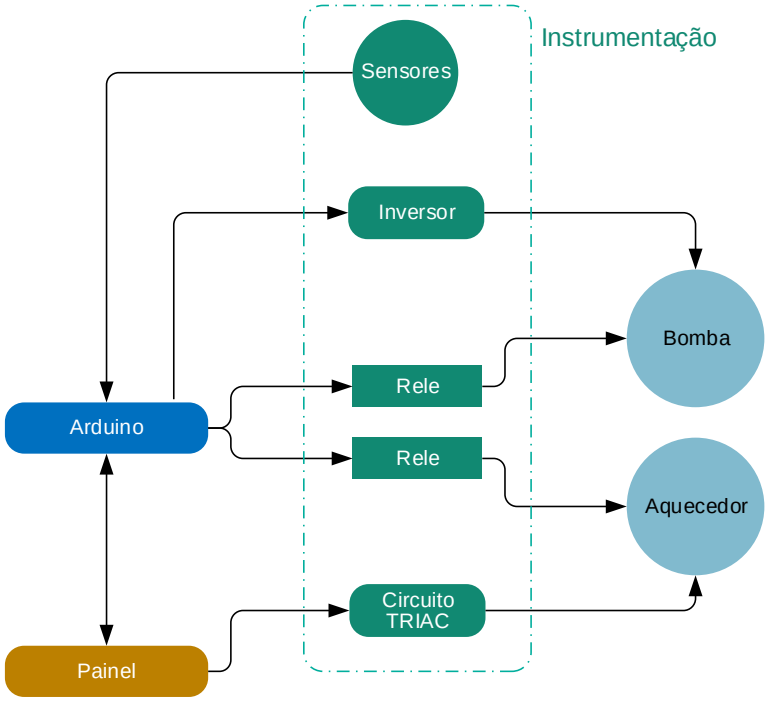

Figura 2.3 – Arquitetura do sistema instalado

2.2 Tecnologias para Controle e Monitoramento de Processos

A seguir são apresentadas algumas tecnologias utilizadas para implementar sistemas de controle e monitoramento de processos.

2.2.1 Sistemas SCADA

Aplicações SCADA são utilizadas na supervisão e controle de largamente empregadas na indústria em setores como saneamento, energia, metalurgia, manufatura entre outros (POLÔNIA, 2011). Estes sistemas coletam dados de dispositivos de campo e os concentram em ambientes onde possam ser processados e armazenados. Os operadores podem visualizar esses dados através de interfaces gráficas (IHMs), que ilustram o que está ocorrendo na planta em tempo real.

A arquitetura de um sistema SCADA é mostrada na Figura 2.4. Estes sistemas são compostos por componentes que são definidos na literatura de acordo com a contribuição no processo de coleta, processamento e exibição dos dados de uma planta.

Dispositivos de campos podem ser entendidos pelos sensores e atuadores. Os sensores medem grandezas físicas do processo. enquanto os atuadores interferem no processo como por exemplo através acionamento de bombas e válvulas.

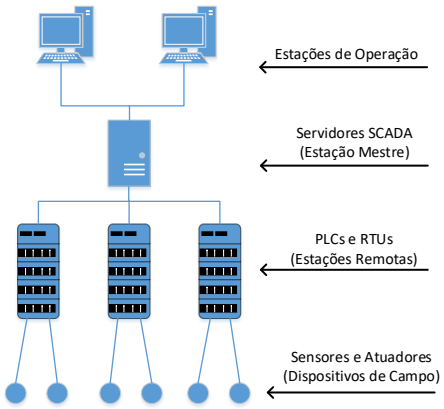

Figura 2.4 – Arquitetura do sistema instalado. Adaptado de Polônia (2011)

As estações remotas são dispositivos programáveis que possuem a função de fazer a interface entre dispositivos de campo e estações mestre, além de serem dotados de capacidade de executar algoritmos de controle local. Nessa categoria enquadram-se os CLPs e RTUs. Os CLPs surgiram no contexto de chão de fábrica, ou seja, necessidade de programação e menor alcance de comunicação, devida a pequena distância entre os equipamentos. As RTUs surgiram da necessidade de enviar dados de sistemas localizados a grandes distâncias da estação mestre. Contudo, a evolução desses equipamentos tornou difícil a distinção entre eles (BAILEY; WRIGHT, 2003).

As estações mestres são o cérebro do sistema. Elas são responsáveis por solicitar dados das estações remotas, processá-los, armazená-los, bem como repassar comandos dos operadores para as estações remotas. As estações mestres são computadores, cuja capacidade depende do porte do projeto. Existem diferentes fabricantes de software que podem ser executados nesses servidores. Alguns fabricantes disponibilizam a opção de separar o processamento em diferentes estações mestres.

As estações de operação executam um software que apresenta uma interface gráfica para o usuário, em que o operador consegue visualizar e interferir no processo através de controles (botões, sliders, etc) disponibilizados pela interface. Em pequenas aplicações, as estações de operação e as estações mestres se encontram nos mesmos computadores.

A comunicação entre estação mestre e remota se dá pela utilização de protocolos industriais abertos como por exemplo Modbus[7], DNP3[8], OPC[9], entre outros; também há a comunicação através de protocolos proprietários. Este último caso ocorre em sua maioria quando a estação mestre e remota são fornecidos pelo mesmo fabricante.

7 http://www.modbus.org/
8 https://www.dnp.org/Default.aspx
9 https://opcfoundation.org/about/what-is-opc/

2.2.2 Tecnologia Embarcada

Como mencionado no Capítulo 1, os sistemas embarcados, estão cada vez mais presentes no setor industrial. Malinowski e Yu (2011), fazem uma pesquisa sobre os modelos de sistemas embarcados e em quais tipos de aplicações industriais estão sendo utilizados.

Destacam-se a utilização de microcontroladores para executar algoritmos de controle e comunicação com outros sistemas. Um estudo de caso cita o túnel de vento instalado na Universidade de Bradley. A arquitetura do sistema implantado é mostrada na Figura 2.5. Comparando com a arquiteura SCADA descrita anteriormente, os elementos marcados com o número 1, seriam os dispositivos de campo, o microcontrolador (2), seria a estação remota, o PC executando o LabView (3) seria a estação master, e o outro computador seria a estação de visualização (IHM).

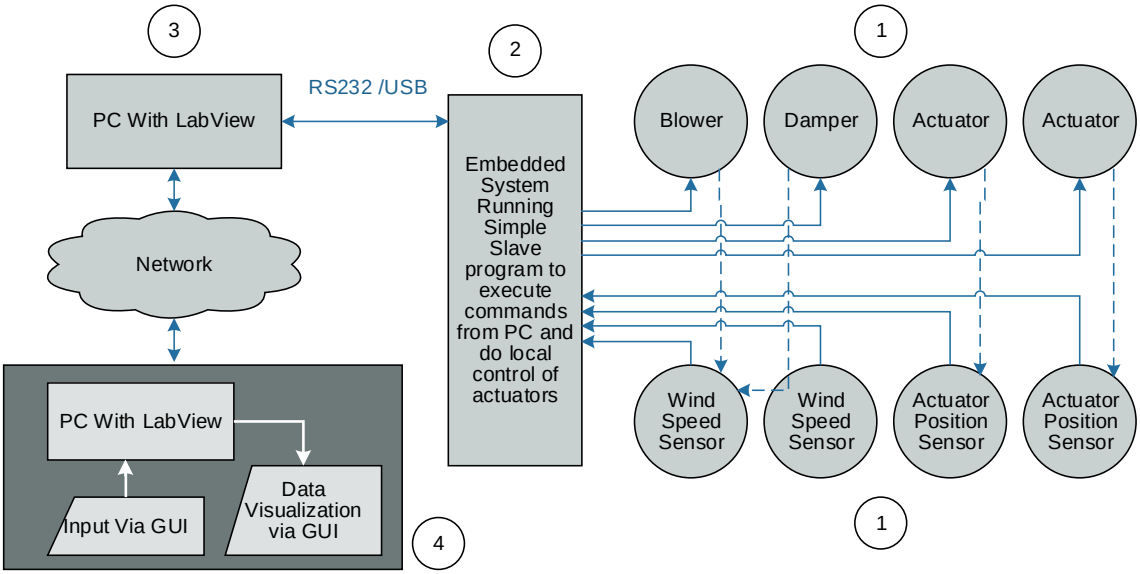

Figura 2.5 – Exemplo de Sistema com tecnologia embarcada. Adaptado de Malinowski e Yu (2011)

O estudo ainda menciona sobre crescimento da utilização de FPGAs[10] em funções ocupadas pelos microcontroladores, devido a redução dos custos de aquisição dos mesmos, bem como pelas melhorias observadas em relação a capacidade de reconfiguração, o que facilita a sua utilização.

É possível encontrar também diversos trabalhos na literatura que utilizam plataformas de prototipagem eletrônica para conceber projetos de automação para o setor residencial, para plantas de pequeno porte e para projetos na área educacional como plantas didáticas. A portabilidade é uma característica desejada para aplicações desses setores, que também são caracterizadas por possuírem restrições de custo, o que motiva a utilização de hardwares e softwares livres bem como ferramentas multiplataforma.

[10] FPGAs e Microcontroladores: https://www.7pcb.com/blog/fpgas-and-microcontrollers/

2.2.3 Plataformas de Prototipagem

O termo prototipagem rápida designa um conjunto de tecnologias utilizadas para fabricação de objetos físicos diretamente a partir de fontes de dados gerados por sistemas de projeto auxiliado por computador (GORNI, 2001). A construção do objeto através da agregação de camadas permitem aos projetistas criar protótipos rapidamente.

Este conceito pode ser extrapolado para o desenvolvimento dispositivos eletrônicos. Atualmente no mercado, encontram-se placas baseadas em microcontroladores e microprocessadores que podem se agregar a módulos, construindo assim, um novo produto. Essas placas são reprogramáveis, o que torna o processo de prototipagem rápido e flexível. A seguir serão apresentadas duas das principais placas de prototipagem do mercado.

2.2.3.1 Arduino

Arduino é uma plataforma eletrônica open-source baseado em hardware e sofware flexíveis e fáceis de usar (ARDUINO, 2017). Basicamente a plataforma Arduino consiste em um conjunto de placas que possuem um microcontrolador, terminais de entradas e saídas analógicas, digitais e de dados, e uma porta para programação. Como se trata de um hardware livre, é possível que os usuários produzam as próprias placas.

As placas se diferem pela arquitetura do microcontrolador, número de terminais, memória, entre outros. Uma placa pode ter as suas funcionalidades aumentadas através da inserção de Shields, que nada mais são que outras placas fabricadas com propósitos específicos. A placa Arduino UNO[11] é uma das mais conhecidas, porém existem diversas placas mais poderosas, inclusive que suportam sistema operacional linux, como o Arduino Yun[12]. Como mencionado anteriormente, foi utilizado o Arduino DUE no projeto, que está categorizada como uma das placas mais potentes e com maior número de terminais.

Essa plataforma conta com inúmeras bibliotecas desenvolvidas pela comunidade usuária, que podem ser baixadas diretamente da página do Arduino, ou então disponibilizadas em páginas de compartilhamento de código open-source como o GitHub. A página do Arduino também disponibiliza uma IDE (Integrated Development Environment) para programação da placa. Se o usuário deseja programar em um nível mais baixo, como por exemplo acesso aos registradores internos do Arduino, é possível utilizar o Atmel Studio[13].

2.2.3.2 Raspberry PI

Um Raspberry PI é um computador de pequena dimensão (comparável com a dimensão de um cartão de crédito). O objetivo inicial da criação era criar um dispositivo de

[11] https://store.arduino.cc/usa/arduino-uno-rev3
[12] https://store.arduino.cc/usa/arduino-yun
[13] https://www.embarcados.com.br/atmel-studio/

baixo custo para incentivar a prática de programação em níveis anteriores à graduação. Porém devido ao seu pequeno tamanho, baixo custo e razoável poder de processamento foi sendo adotado em muitos projetos eletrônicos. As placas Rasberry aceitam diversos sistemas operacionais, sendo que o sistema operacional oficial é o Raspian[14], uma versão da distribuição Debian/Linux[15]. Ao longo dos anos, foram lançadas algumas versões das placas, que foram aumentando seu poder de processamento e inserindo mais funcionalidades. Um resumo das características das principais versões lançadas pode ser visto na Tabela 2.1. Uma das grandes vantagens da versão 3, que foi a versão adquirida para o projeto é possuir os módulos Bluetooth e Wifi integrados à placa.

Tabela 2.1 – Comparação entre versões do Raspberry.
Adaptado de https://en.wikipedia.org/wiki/Raspberry_Pi

	Raspberry 1	**Raspberry 2**	**Raspberry 3**
Lançamento	02/2013	02/2015	02/2016
Preço	$25	$35	$35
Arquitetura	ARMv6Z	ARMv7-A	ARMv8-A
CPU	700Mhz one core	900Mhz quad-core 64bit	1.2Ghz quad-core 64bit
Memória	256MB	1GB	1GB
Nº de Portas USB	1	4	4
Corrente	300mA	220mA - 820mA	300mA - 1.34A
Wifi Integrado	Não	Não	Sim. 802.11n
Bluetooth Integrado	Não	Não	Sim. v4.1

O Raspberry possui um conjunto de pinos, que são úteis para realizar comunicação com outros dispositivos através de protocolos como UART, SPI e I2C, bem como enviar e receber informações digitais (nível lógico 0 ou 1) de dispositivos diversos. A Figura 2.6 mostra os componentes da placa Raspberry Pi 3, bem como uma breve explicação das funcionalidades dos 40 pinos. Observa-se que os pinos que são capazes de realizar comunicação são específicos. Outra informação importante é que os pinos lógicos suportam níveis de tensão até 3.3v, o que difere da tecnologia TTL tradicional, que suporta 5V. Ou seja, caso os pinos sejam submetidos a tensões de 5V, corre-se o risco de danificar a placa.

2.2.3.3 Protocolos de Comunicação

Um protocolo de comunicação consiste em um conjunto de regras e convenções criadas para possibilitar a comunicação entre dispositivos. Alguns exemplos de regras existentes em um protocolo: como um dispositivo identifica o outro, quando pode enviar

[14] https://www.raspberrypi.org/downloads/raspbian
[15] https://www.debian.org/

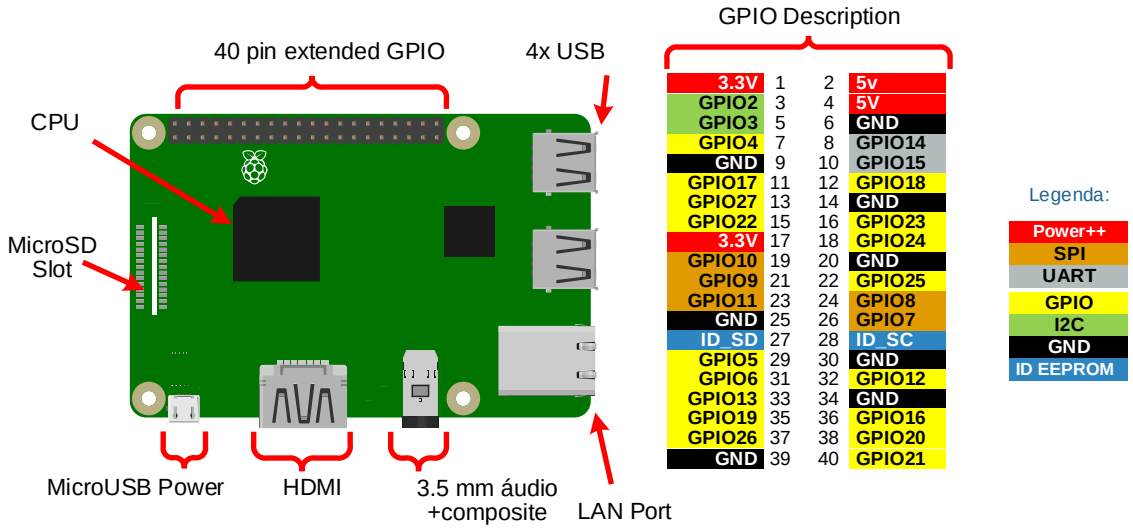

Figura 2.6 – Composição de um Raspberry PI 3

e receber a informação, quanto tempo limite de espera por uma mensagem, entre outros (RODRIGUES; FARIA, 2016).

Existem alguns protocolos para estabelecer a comunicação entre Raspberry e Arduino. Basicamente, os protocolos são implementados por bibliotecas, que podem ser utilizadas por Raspberry e Arduino. Como é possível criar softwares em várias linguagens no Raspberry, as bibliotecas a serem utilizadas para esse dispositivo dependem da linguagem de programação utilizada.

Como mostrado na Figura 2.6, o Raspberry possui terminais para comunicação SPI, I2C, e UART. Estes protocolos também são suportados pelo Arduino. Robocore (2017) e Breseman (2015) apresentam um comparativo entre esses protocolos, cujas caracterísitcas estão descritas na Tabela 2.2

Tabela 2.2 – Comparação entre SPI I2C e UART. Adaptado de Robocore (2017)

Protocolo	Taxa (bps)	Sentido Transmissão	Método	Nº Fios	Tensão	Max dispositivos
UART	1200 a 115200	Full-Duplex	Assíncrono	2	0 a 5V	1
SPI	0 a 10mb	Full-Duplex	Síncrono	3+N	0 a 5V	não há
I2C	100k ou 400k	Half-Duplex	Síncrono	2	0 a 5V	127 ou 1024

O protocolo SPI possui a vantagem de o mais rápido entre eles e também ser Full-Duplex. As desvantagens consistem em: aumento do número de fios à medida que o número de dispositivos na rede cresce; alguns dispositivos pode não suportar velocidades muito altas, o que pode causar incompatibilidade na comunicação; é mais suscetível a ruídos.

O protocolo UART possui como vantagem a simplicidade das mensagens envolvidas na comunicação, bem como a sua disponibilidade em diversos dispositivos. Muitas vezes utilizam-se USB e RS-232 para trafegar dados em UART. Como desvantagem destaca-se: limitação em termos de velocidade; ser assíncrono, ou seja é necessário que a taxa de transmissão e recepção seja a mesma para ambos os dispositivos.

O I2C se situa em uma região intermediária entre UART e SPI. É um protocolo síncrono, utiliza-se de apenas 2 fios, e possui uma taxa de transmissão um pouco maior que o UART. Como desvantagem, destaca-se a necessidade de interpretar os pacotes de dados via software, o que pode diminuir a performance de processamento.

2.2.3.4 Exemplos de Projetos que utilizaram Plataformas de Prototipagem

Encontra-se na literatura, projetos que utilizam placas de prototipagem mencionadas acima, para implementar um sistema de automação de pequenas instalações.

Rodrigues e Faria (2016) implementou a automação de uma planta experimental de um sistema de reaproveitamento de água. O modelo experimental foi construído, instalaram-se sensores e atuadores que se conectam a Arduino. O Arduino se comunica com um Raspberry PI, que possui um sofware SCADA instalado, o Mango automation. Ou seja, essa arquitetura é próxima a de um sistema SCADA tradicional, mostrada na Figura 2.4. O Arduino faz o papel de uma estação remota, e o Raspberry, de uma estação mestre.

Existe um projeto open-source de um sistema de controle de temperaturas para cervejarias chamado BrewPi (BREWPI, 2017). O sistema consiste em um Raspberry PI que comunica com um Arduino, que é encapsulado a uma caixa que possui um display LCD para visualização de informações em modo local. O Arduino implementa algoritmos de controle em malha fechada, bem como possui funcionalidades para ler informações do processo, bem como acionar aquecedor e refrigerador. O Raspberry implementa um WebServer, que disponibiliza os dados do processo para os usuários. Para monitorar o processo, basta um dispositivo que possua um navegador Web. Fazendo um comparativo com a arquitetura SCADA tradicional, o WebServer seria o software SCADA que é executado nas estações mestres, e as estações de operação podem ser representadas por qualquer dispositivo com um navegador Web. Essa arquitetura concede portabilidade à operação, característica desejável a qualquer projeto. A arquitetura do sistema Brewpi é exibida na Figura 2.7. Este projeto é um dos exemplos do crescimento da utilização de sistemas Web para monitoramento e controle de processos.

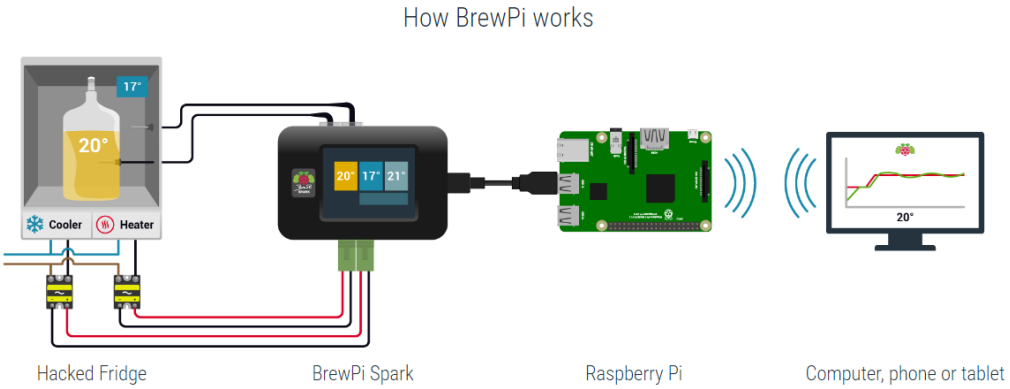

Figura 2.7 – Arquitetura do sistema BrewPi. Fonte: https://www.brewpi.com/

2.3 Aplicações Web

A Web, abreviação do termo *World Wide Web*, que significa em tradução livre "rede de alcançe mundial", também conhecida como www, é um sistema distribuído de hipermídia que revolucionou como pessoas organizações e sistemas geram, acessam e compartilham informações (LEE, 2000). Foi criada em 1989 por Tim Berners Lee resolver problemas em relação a transmissão e compartilhamento de informações do CERN (Organização Europeia para a pesquisa Nuclear).

Porém o sucesso da Web fez com que a sua utilização não ficasse restrita ao meio acadêmico, sendo utilizada para fins pessoais e comerciais. O crescimento exponencial da Web causou um temor entre a comunidade e pesquisadores da internet, sobre a estabilidade da rede diante da crescente demanda. Diante disso, surgia a necessidade de melhorar a Web em termos de escalonabilidade performance e padronização em relação as aplicações. Em 1994 Berners-Lee fundou a W3C, órgão responsável pela padronização de tecnologias Web, instituição mais importante e reconhecida no mundo sobre o assunto (LALLI; BUENO; ZACHARIAS, 2008).

Ao longo dos anos, a Web sofreu transformações, em relação a aplicabilidade e a forma de transmitir informações. Inicialmente, as páginas Web publicavam apenas conteúdo estático, ou seja textos sem muita interatividade. O crescimento da Internet e o aumento da largura de banda fez com que fossem criados sites cada vez mais dinâmicos e interativos, incluindo a participação do internauta na geração de conteúdo, como por exemplo as redes sociais. Atualmente sistemas complexos que antes eram desenvolvidos apenas como aplicativos Desktop, podem ser criados utilizando somente tecnologias Web. A Figura 2.8 e a Figura 2.9 exibem um resumo da evolução da web, suas tecnologias e aplicações.

Uma aplicação Web tradicional é baseada em estilo cliente-servidor. Basicamente, um servidor oferece serviços e espera por requisições, enviadas por clientes interessados

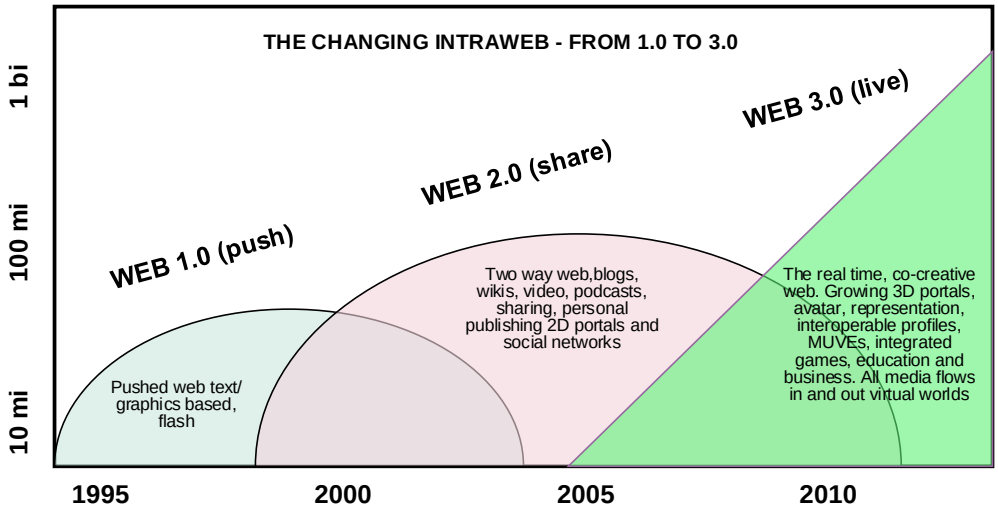

Figura 2.8 – Evolução da Web. Adaptado de http://www.personalizemedia.com/virtual-worlds-web-30-and-portable-profiles/

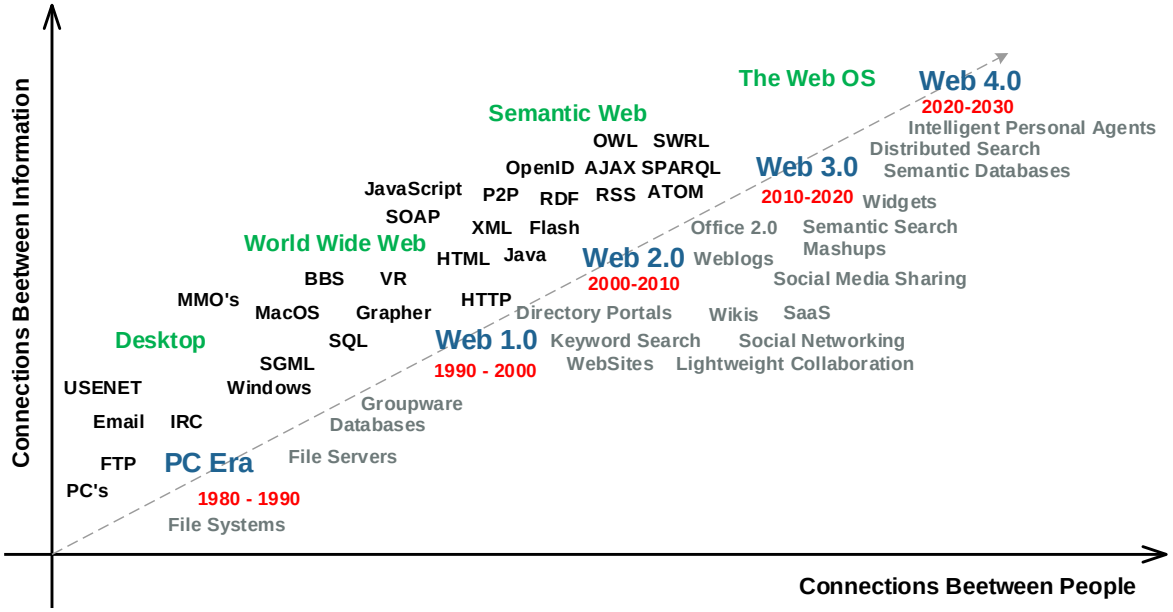

Figura 2.9 – Evolução da Web. Tecnologias e Aplicações. Adaptado de https://www.prokarma.com/blog/2014/10/16/what-exactly-web-30

nesses serviços. Ao receberem as requisições, os servidores processam a requisição e enviam uma resposta ao cliente, que é representado por navegadores web. A Figura 2.11 mostra o fluxo de informações de uma requisição web e a resposta do servidor. As informações trafegam pelo protocolo HTTP[16]. Um frame HTTP de requisição possui a estrutura mostrada na Figura 2.10. O método GET é o mais utilizado. Ele possui a função de recuperar algum recurso, que é identificado por sua URL, do servidor Web. Existem outros métodos

[16] Especificação HTTP: https://www.ietf.org/rfc/rfc2616.txt

no protocolo HTTP, como POST, PUT, DELETE, etc (MACEDO, 2016). O conteúdo a ser exibido nas páginas é produzido e armazenado em arquivos HTML, que basicamente é um arquivo de texto com marcações. Os navegadores interpretam os arquivos e exibem a página formatada para o usuário. Diante da característica dinâmica das páginas web atuais, outros arquivos se fazem necessários como arquivos JavaScript e arquivos CSS, que são arquivos processados nos navegadores.

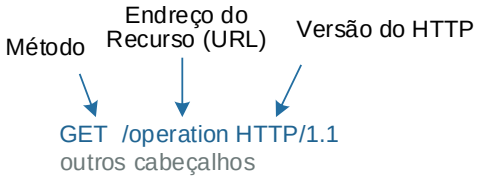

Figura 2.10 – Frame de requisição HTTP

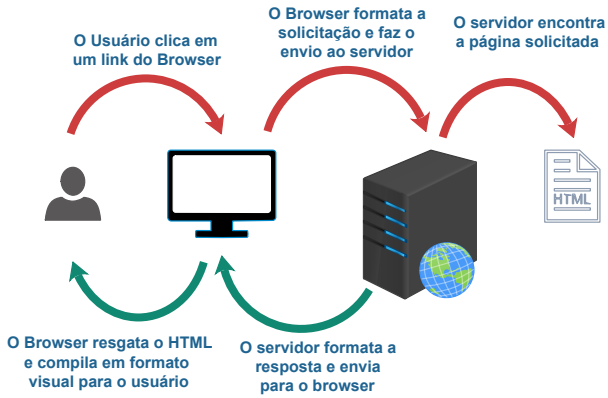

Figura 2.11 – Funcionamento do sistema requisição resposta. Adaptado de https://www.devmedia.com.br/como-funcionam-as-aplicacoes-web/25888

2.3.1 Frameworks Web

Um Framework Web é um framework[17] de software, designado para facilitar o desenvolvimento de aplicações web. Possui funções para automatizar tarefas comuns em aplicações web, como por exemplo acesso à banco de dados, gerenciamento de sessões de conexões, geração templates para geração de HTML, e mapeamento de rotas. Sendo assim, a utilização de um framework web torna o desenvolvimento de uma aplicação mais rápido e fácil. Existem frameworks Web para as mais diversas linguagens de programação, com características e funcionalidades distintas. A escolha do framework é subjetiva: depende do propósito da aplicação, conhecimento do desenvolvedor, entre outros fatores. A Figura 2.12

[17] Definição de framework: https://pt.wikipedia.org/wiki/Framework

mostra um ranking dos Frameworks mais populares, baseado na avaliação do framework no Github, e no número de questões do mesmo presentes no StackOverflow.

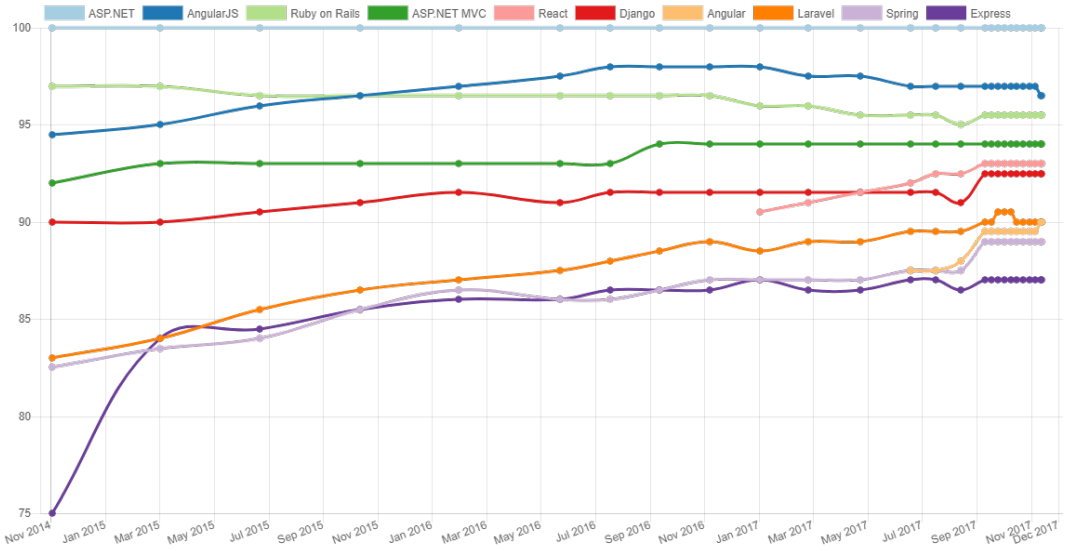

Figura 2.12 – Frameworks Web mais populares. Fonte: https://hotframeworks.com/

2.3.1.1 Django

Django é um framework web desenvolvido na linguagem Python. É o framework mais utilizado escrito em Python, e um dos mais utilizados entre todas as linguagens (ver Figura 2.12). Possui como características principais (DEVELOPERS, 2017):

1. Completude: O Django provê todas as funcionalidades básicas de uma aplicação web: compatibilidade com a maioria dos bancos de dados, recursos de proteção e segurança, controle de seções e cache,entre outros.

2. Versatilidade: O framework pode ser utilizado para desenvolver qualquer tipo de website. Consegue ser hospedado em vários servidores Web, bem como consegue trabalhar com qualquer framework para o lado cliente, como por exemplo o jquery[18].

3. Portabilidade: Como é escrito em Python, uma aplicação desenvolvida em Django pode ser hospedada nos principais sistemas operacionais (Linux, MacOS, Windows).

4. Manutenibilidade: O framework foi concebido de acordo com princípios que incentivam a criação de códigos reutilizáveis e de fácil manutenção. Por exemplo, o django disponibiliza estruturas para implementar o DRY (Don't Repeat Yourself) filosofia que reduz a necessidade de replicação de código.

[18] https://jquery.com/

5. Escalabilidade: Um projeto desenvolvido em Django é composto de várias aplicações, que funcionam de forma independente entre si dentro do projeto. Essa separação entre partes do projeto permite além de flexibilidade (uma parte pode ser substituída ou modificada sem afetar o restante do código), crescimento escalável.

6. Mapeamento Objeto-Relacional (ORM): É uma técnica que permite manipular dados de banco usando paradigma orientado a objetos. Basicamente, um modelo de dados é mapeado como uma tabela no banco de dados. Dessa forma elimina-se a necessidade de escrever códigos SQL para manipulação de bancos.

7. Popularidade: Possui uma extensa documentação e uma grande comunidade (em fóruns) disposta a ajudar.

Como Django foi feito para possibilitar desenvolvimento rápido, muitas decisões de características do projeto fica a cargo do framework, o que é uma desvantagem. Outro ponto negativo é a geração de muitos arquivos mesmo para pequenos projetos.

Um projeto em Django é criado sobre o padrão MVC[19], que em resumo é uma forma de programação que separa a representação da informação, do processamento dos dados em si. Dessa forma é possível possui por exemplo várias representações de uma mesma informação. A estrutura de uma aplicação em Django é exibida na Figura 2.13.

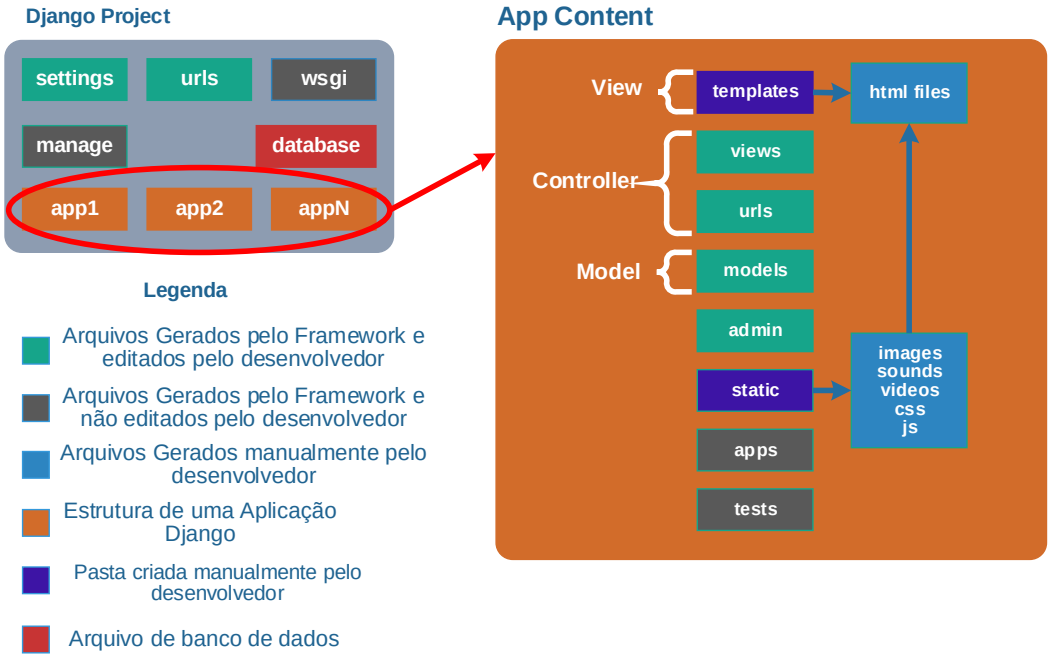

Figura 2.13 – Estrutura de um projeto Django

[19] Padrão MVC: https://developer.chrome.com/apps/app_frameworks

3 Desenvolvimento

Este capítulo apresenta o desenvolvimento do sistema objetivo deste trabalho. São apresentadas as decisões de projeto, arquitetura e alguns trechos não triviais do códigos são explicados

3.1 Decisões de Projeto

O estado atual do trocador de calor foi descrito na **??**. O intuito deste projeto é implementar um sistema de monitoramento e controle remoto sem que haja alteração da infraestrutura já instalada. Em resumo, o objetivo é incluir algum dispositivo que necessite apenas se comunicar com o Arduino para executar suas tarefas.

Para este projeto, é possível utilizar tanto uma arquitetura SCADA tradicional, quanto tecnologias embarcadas como placas de prototipagem. Foi definido utilizar a segunda opção, devido a fatores econômicos e também pela portabilidade concedida ao sistema. A arquitetura foi inspirada em grande parte no sistema BrewPi, descrito na **??**. O Raspberry Pi 3 foi escolhido por possuir comunicação Wifi integrada à placa, e possuir uma documentação mais extensa que outras placas similares no mercado como o BeagleBone Black[1] e a recente e também poderosa placa DragonBoard 410c[2]. Dessa forma, a visão geral do sistema proposto é exibida na Figura 3.1.

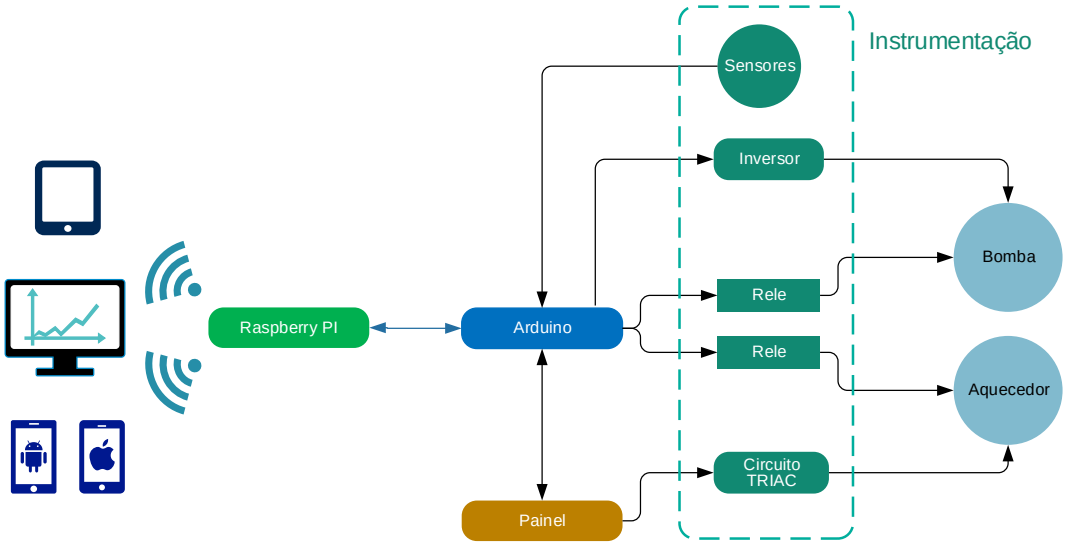

Figura 3.1 – Nova Arquitetura Proposta para o Sistema

[1] https://beagleboard.org/black
[2] https://developer.qualcomm.com/hardware/dragonboard-410c

3.2 Requisitos Funcionais do Sistema

Uma vez definido o hardware a ser utilizado, deve-se levantar os requisitos funcionais do sistema, ou seja o que o sistema deve fazer e como deve reagir em determinadas situações. Basicamente, os requisitos funcionais são as características de operação do sistema de monitoramento a ser desenvolvido. Os requisitos estão resumidos na Tabela 3.1.

Tabela 3.1 – Requisitos Funcionais do Sistema

Índice Requisito	**Descrição Requisito**
1	Exibir a informação atual das variáveis de processo (vazões e temperaturas)
2	Exibir o estado da bomba e do aquecedor (Ligado ou Desligado)
3	Exibir o valor da rotação atual da bomba
4	Em modo remoto, deve permitir que o operador acione a bomba e o aquecedor
5	Em modo remoto, deve permitir que o operador altere a velocidade da bomba
6	Permitir a visualização dos dados analógicos em gráficos
7	Impedir a atuação do operador quando o Arduino estiver em modo local ou de emergência
8	Armazenar os dados do sistema em banco de dados
9	Permitir que o usuário habilite ou desabilite o armazenamento de dados das variáveis
10	Permitir que o usuário colete as informações contidas no banco em um arquivo no formato csv
11	Permitir que o usuário apague as informações contidas no banco de dados

3.2.1 Descrição Funcional

Para atender aos requisitos citados acima, optou-se por modularizar as funcionalidades do sistema. Dessa forma, foram definidos 3 componentes. O termo componente deve ser entendido como um programa. Os componentes foram projetados de forma que cada um funcione de forma mais independente possível. Assim, o impacto no sistema caso haja alguma alteração de componente, em termos de reengenharia, é minimizado.

A relação entre os componentes é exibida na Figura 3.2. Foram definidos 3 componentes: o programa que é executado no Arduino; o Gateway e o WebServer, que são programas executados no Raspberry. Optou-se por criar dois componentes no Raspberry

PI, de forma que as funcionalidades do sistema fossem corretamente agrupadas. As atribuições de cada programa estão resumidas na Tabela 3.2.

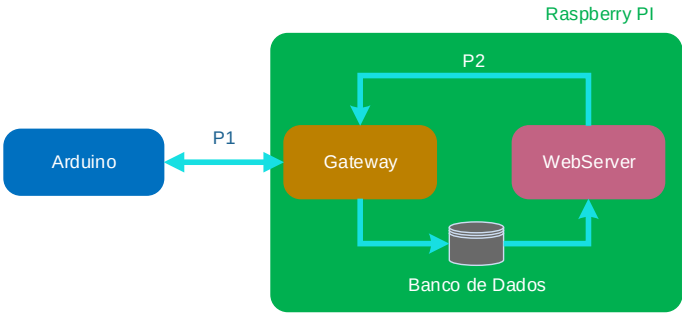

Figura 3.2 – Arquitetura Detalhada

Tabela 3.2 – Atribuições de cada Componente

	Atribuições
Arduino	1 - Interface com sensores e Atudadores 2 - Interface com Painel 3 - Enviar estado do sistema quando solicitado pelo Gateway 4 - Interpretar comandos enviados pelo Gateway
Gateway	5 - Solicitar de forma cíclica o estado das variáveis 6 - Armazenar os dados recebidos em um banco de dados 7 - Receber comandos vêm do WebServer (comandos dos usuários) e repassar ao Arduino
WebServer	8 - Ler dados do banco e apresentar ao usuário 9 - Receber comandos do usuário e repassar ao Gateway

É importante observar que o Gateway apenas cuida da transferência de informações entre WebServer e Gateway. O programa contém duas threads: uma para realizar a solicitação cíclica de dados ao Arduino e armazená-los no banco de dados; e a outra para aguardar o envio de comandos vindos do WebServer e repassar ao Arduino. A comunicação entre Gateway e Arduino é realizada através do protocolo I2C, e a comunicação entre Gateway e WebServer é feita através de sockets TCP/IP.

O sistema foi projetado de forma que a substituição de um componente não influencie os demais, contudo alterações nos protocolos influenciaria o código dos componentes envolvidos. Portanto os protocolos devem ser mantidos.

3.3 Implementação

Essa seção consiste em detalhar o funcionamento dos componentes e a comunicação entre eles. Alguns trechos de códigos, considerados não triviais são explicados.

3.3.1 Arduino

O código fonte completo está disponibilizado no Github[3], no link https://github.com/felipefonsecabh/ArduinoCode/blob/ArduinoNoNavigation/ArduinoCode.ino

Foi feita uma análise do programa atual que o Arduino executa. O programa é extenso (contém 1482 linhas), sendo que cerca de 90% do código é dedicado a cuidar do sistema de navegação do display LCD através dos botões existentes no painel. A implementação de um sistema de monitoramento elimina a necessidade dessa estrutura de navegação, de modo que o painel passe a ter apenas funcionalidades diretas de acionamento, e o display exiba apenas informações básicas. Portanto, a melhor opção foi reformular o código do Arduino e aproveitar apenas as funções que fazem a leitura dos sensores e convertem os valores para unidade de engenharia. Essas funções se fazem necessária, pois não é escopo do projeto recalibrar sensores e/ou alterar a instrumentação já existente.

A estrutura do código, que está exibida na Figura 3.3, foi montada de forma a permitir rápida adaptação do mesmo para diferentes protocolos de comunicação, e está de acordo com as atribuições mencionadas na Tabela 3.2. As trocas de mensagens com o Gateway são feitas por interrupção, ou seja, estão fora da função *loop* do Arduino.

O programa foi intensamente modularizado. Todos os estados e funcionalidades foram encapsuladas em funções. Esse tipo de prática acelera a interpretação do código. Outro benefício é a possibilidade de expansão de funcionalidades, como por exemplo a implementação de malhas de controle no código.

No diagrama é utilizado o termo "estado do sistema", que deve ser interpretado como o conjunto de variáveis necessários para descrever o sistema. Essas variáveis são: as 4 temperaturas; 2 vazões; estado de acionamento de bomba e aquecedor (Ligado ou Desligado); velocidade da bomba; modo de operação (Local ou Remoto) e se o botão de emergência está acionado ou não.

Para a leitura dos sensores foram utilizadas bibliotecas desenvolvidas e mantidas por terceiros. A Tabela 3.3 mostra a relação entre tabelas, desenvolvedores e versões. Não foi necessária nenhuma biblioteca para a leitura da vazão de água fria. O sensor de vazão consiste em um dispositivo que envia pulsos ao Arduino. Quanto mais pulsos enviados, maior é a vazão. Portanto, é feita uma contagem de pulsos em um intervalo de tempo fixo, e posteriormente esse número é inserido em uma fórmula que retorna o valor da vazão. A

[3] https://www.oficinadanet.com.br/post/14791-o-que-github

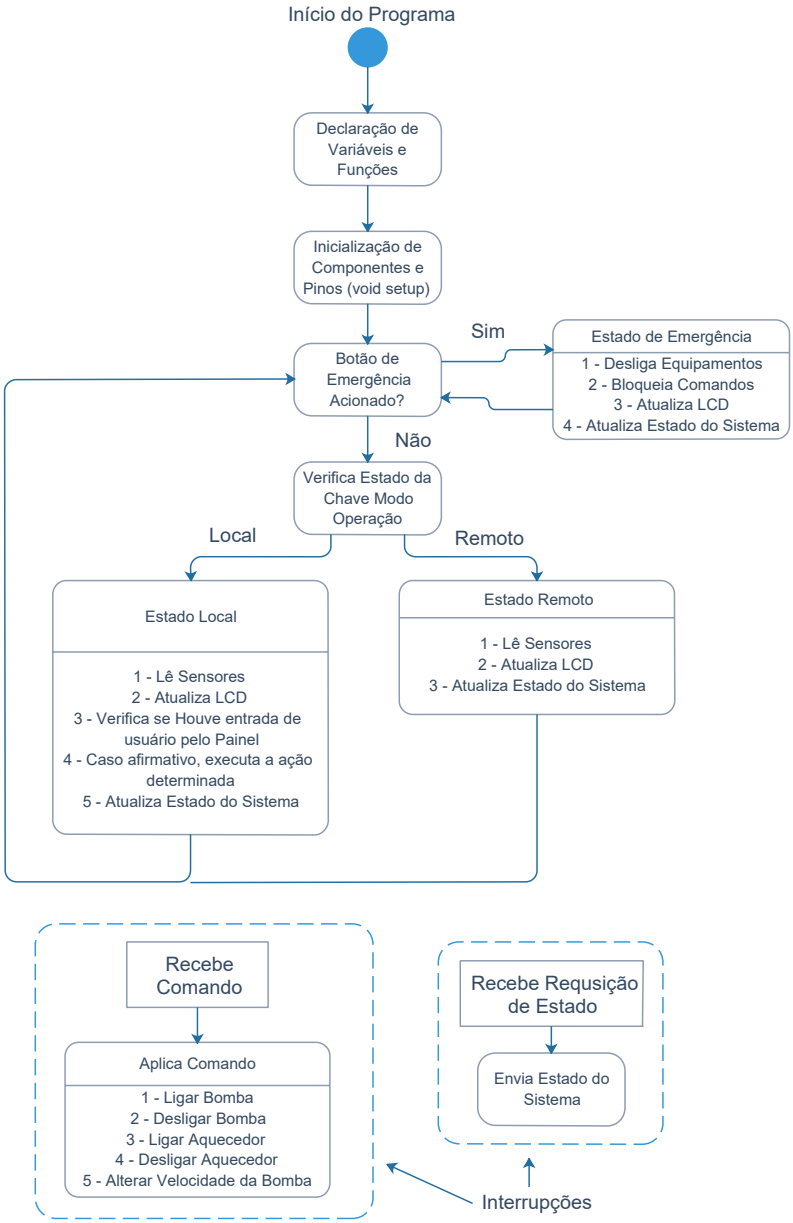

Figura 3.3 – Estrutura do Código do Arduino

contagem de pulsos é feita por interrupção.

O código correspondente a leitura dos valores analógicos é mostrado no Listing 3.7.

A troca de dados entre Arduino e Gateway é feita pelo protocolo I2C. Este protocolo é implementado no Arduino através da biblioteca Wire. Neste projeto o Arduino atua como slave da comunicação, ou seja, não inicia a comunicação. Responde a algum dispositivo mestre apenas quando solicitado.

Quando o mestre envia um comando de escrita, atua-se uma interrupção, que

Tabela 3.3 – Bibliotecas utilizadas no programa do Arduino

Biblioteca	**Versão**	**Mantida por**	**Função**
Dallas Temperature	v3.7.6	Burton (2016)	Temperatura
OneWire	v2.3.3	Stoffregen (2017)	Temperatura
Ultrasonic	-	FilipeFlop (2015)	Vazão Quente

```
void Temperaturas() {
  // call sensors.requestTemperatures() to issue a global temperature
  // request to all devices on the bus
  sensors.requestTemperatures();

  // print the device information
  for (byte i = 0; i <= 4; i++){
    temp[i] = sensors.getTempC(deviceID[i]);
  }
}

void VazaoAguaFria(){
  currentTime = millis();
  // Every second, calculate litres/hour
  if (currentTime >= (cloopTime + 1000)){
    cloopTime = currentTime; // Updates cloopTime
    // Pulse frequency (Hz) = 7.5Q, Q is flow rate in L/min.
    vazao_fria = (flow_frequency / 7.5); // (Pulse frequency) / 7.5Q =
    ↪  flowrate in L/min
    flow_frequency = 0; // Reset Counter
  }
}

void VazaoAguaQuente(){
  float vazao1_sf; //descobrir o porque do nome da variavel
  microsec = ultrasonic.timing();
  cmMsec = ultrasonic.convert(microsec, Ultrasonic::CM);
  nivel = 11.46 - cmMsec;
  vazao1_sf = (0.0537)* pow((nivel * 10), 1.4727);
  if (vazao1_sf > 1){
    vazao_quente = 0.75*vazao_quente + 0.25*vazao1_sf;
  }
}
```

Listing 3.1 – Funções de Leitura dos sensores

executa a função onReceive. Quando o mestre envia uma solicitação de dados, atua-se uma outra interrupção, que executa a função onReceive e posteriormente executa a função onRequest. Em suma, a função onReceive é utilizada para interpretar comandos, como por exemplo ligar ou desligar bomba e aquecedor, e a função onRequest é utilizada para enviar para o mestre dados sobre o processo como por exemplo valores de temperaturas e vazões.

As informações no protocolo I2C trafegam sob a forma de um array de bytes.

Portanto, as informações referentes ao sistema, que são booleanas e reais, devem ser convertidas em um array de bytes para serem transmitidas. É possível realizar essa conversão utilizando a declaração union. Essa técnica permite que variáveis de diferentes tipos ocupem uma mesma região de memória. Assim, foi criada uma estrutura de dados para representar o estado do sistema, que compartilha essa região de memória com um array de bytes. O Listing 3.2 corresponde a essa implementação feita.

```
//estrutura de dados para envio i2c
typedef struct processData{
  float temp1;
  float temp2;
  float temp3;
  float temp4;
  float hotflow;
  float coldflow;
  float pump_speed;
  byte bstatus;
  byte chksum;
};

typedef union I2C_Send{ //compartilha a mesma área de memória
  processData data;
  byte I2C_packet[sizeof(processData)];
};
```

Listing 3.2 – Estrutura de dados do sistema

Em relação ao recebimento de comandos pelo Gateway, primeiramente é necessário entender como é a estrutura de um frame enviado pelo mestre. A estrutura do frame está descrita na Figura 3.4.

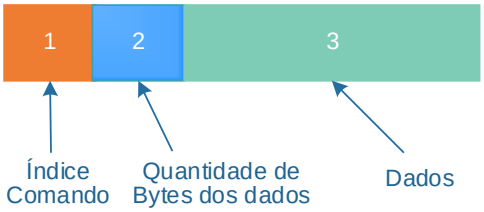

Figura 3.4 – Estrutura de um pacote I2C

O índice de comando arbitrário e é definido pelo desenvolvedor. Portanto, para interpretar corretamente os comandos pelo mestre, foi necessário mapear ações de acordo com o índice de comando criado, ou seja, definir um protocolo básico de comunicação. A relação entre o índice de comando e a ação correspondente é mostrada na Tabela 3.4. A interpretação e execução dos comandos é feita na função onReceive. Se o comando é uma solicitação de dados, o envio é realizado na função onRequest.

Foi colocada a representação decimal e em char dos comandos, porque no Arduíno o comando é interpretado na forma decimal, enquanto no Gateway os comandos são

Tabela 3.4 – Definição das interpretações de comando

Comando (Char)	**Comando (Decimal)**	**Ação**
'1'	49	Ligar a bomba
'2'	50	Desligar a bomba
'3'	51	Ligar aquecedor
'4'	52	Desligar o aquecedor
'5'	53	Alterar velocidade da bomba
'6'	54	Enviar dados do sistema para o mestre

interpretados como char. A relação entre o valor em char e decimal é dada pela tabela ASCII[4].

3.3.2 Preparação do Raspberry

3.3.2.1 Instalação do sistema operacional

O Raspberry não possui memória de armazenamento interna. Portanto, é necessário instalar o sistema operacional (SO) em um microUSB, que faz o papel de HD. Foi utilizada o SO "Raspian Stretch With Desktop", disponível no link https://www.raspberrypi.org/downloads/raspbian/. Basicamente, deve-se extrair a imagem baixada para o microUSB. Após esse procedimento o Raspberry Pi já pode ser inicializado.

3.3.2.2 Ativação do protocolo I2C e Conexão com Arduíno

O I2C não vem habilitado por padrão no sistema operacional. É necessário entrar nas configurações do Rpi e habilitá-lo. Os passos para realizar esse procedimento estão descritos por Matt (2014).

Após ativar o protocolo, é necessário conectar o Raspberry com o Arduíno. A ligação é feita através de dois fios, e ainda o fio para interligar o terra. A Tabela 3.5 mostra com os pinos devem ser interligados.

Tabela 3.5 – Ligação I2C entre Raspberry e Arduíno

Raspberry	**Arduino**
Pino 3	Pino 20
Pino 5	Pino 21
Pino 9	Gnd

[4] https://en.wikipedia.org/wiki/ASCII

3.3.2.3 Instalação de Pacotes - Python

O sistema operacional Raspian já possui duas versões instaladas de Python: 2.7 e 3.4. A primeira, mesmo sendo mais antiga ainda é muito utilizada pela comunidade devido à grande quantidade de pacotes desenvolvidos para essa versão; a segunda, é uma das versões mais novas disponíveis para Python.

Os códigos do Gateway e do WebServer foram desenvolvidos com a versão 3.4. Antes do começo do desenvolvimento é necessário instalar os pacotes necessários para executar o projeto. Os pacotes a serem instalados dependem do propósito e característica do sistema a ser desenvolvido. É muito comum desenvolvedores trabalharem em diferentes projetos, portanto a tarefa de gerenciar os pacotes para cada projeto torna-se trabalhosa. Para isso, o Python disponibiliza a instalação de ambientes virtuais na máquina. Ambientes virtuais são diretórios para armazenar os pacotes necessários para determinados projetos. Assim, é possível isolar os arquivos de cada desenvolvimento, facilitando a mudança de um projeto para o outro (KOWALCZYK, 2017). A utilização de ambientes virutais em Python é uma boa prática de programação e foi utilizada.

Para instalar um ambiente virtual na versão python 3, é necessário utilizar os comandos descritos no Listing 3.3. Uma vez instalado, é necessário ativar o ambiente virtual, cujo comando é exibido no Listing 3.4. Após ativado, devem-se instalar os pacotes necessários para o projeto. Os pacotes e as versões instaladas estão listados na Tabela 3.6. Todos os pacotes, com exceção do Smbus, podem ser instalados através do comando exibido no Listing 3.5. É necessário ativar o ambiente virtual antes de efetuar os comandos. Para a instalação do Smbus, é necessário seguir os passos descritos por Pratyaksa (2015).

```
#navegar até a pasta onde se deseja instalar o ambiente virtual
pi@raspberry $ cd pfc_env

#instalar pacote virtual env
pi@raspberry $ pip install virtualenv

#criar um ambiente virtual chamado env
pi@raspberry $ virtual env
```

Listing 3.3 – Comandos para criação de um ambiente virtual

```
#ativar o diretório virtual
pi@raspberry $ source pfc_env/bin/activate

#caso o nome do diretório apareça entre parenteses como abaixo, o diretório foi
↪  ativado com sucesso
(env) pi@raspberry $
```

Listing 3.4 – Comandos para criação de um ambiente virtual

Tabela 3.6 – Pacotes necessários para o projeto

Pacote	Versão
Smbus	v1.9.2
Django	v1.9.2
Pillow	v1.9.2

```
#ativar o diretório virtual
#o comando é pip install NomePacote== Versão. Exemplo:
(env) pi@raspberry $ pip install Django==1.9.6
```

Listing 3.5 – Comando para a instalação de pacotes Python

3.3.3 Gateway

O código fonte completo do gateway está disponibilizado no Github, no link: https://github.com/felipefonsecabh/PFC/blob/PyServeri2c/arduinoserver.py. Conforme mencionado na subseção 3.2.1, o programa consiste em 2 threads que são executadas continuamente, ou seja em loop infinito.

3.3.3.1 Thread 1: Leitura de Dados do Arduino e Escrita no Banco

Para implementar a comunicação I2C no Gateway, utiliza-se a biblioteca Smbus. Primeiramente é necessário inicializar a conexão com o Arduino, referenciando o endereço do mesmo. Esse endereço é arbitrário, e deve possuir o mesmo valor no código do Arduino e no Gateway. Foi escolhido o identificador 12 para o endereço do Arduino.

```
#inicialização do i2c
bus = SMBus(1)
arduinoAddress = 12
```

Listing 3.6 – Inicialização da comunicação I2C

A solicitação de dados para o Arduino foi feita através da utilização da função read_i2c_block_data. É necessário passar como parâmetro o endereço do Arduino, o ínidice do comando (foi definido na Tabela 3.4 que o índice do comando da solicitação de dados era 54), e o número de bytes esperados na resposta. O resultado retorna um array de bytes que deve ser convertido nas variáveis reais de processo. Para realizar essa conversão foi utilizada a função unpack do pacote struct[5]. Um exemplo de pacote recebido do Arduino é mostrada na figuraX.

Portanto, é necessário separar o array de bytes em 7 variáveis float e 2 variáveis do tipo byte. O primeiro byte contém as informações de status digitais, e o segundo byte é

[5] https://docs.python.org/2/library/struct.html

apenas um valor para verificação da integridade do pacote. Após a separação em variáveis, os dados foram organizados em um dicionário[6] para facilitar o envio para o banco.

```
#faz requisicao pelos dados
block = bus.read_i2c_block_data(arduinoAddress,54,30)
#conversao do array de bytes em variaveis
data = unpack('7f2b',bytes(block))
#inserção dos dados em um dicionário
bstatus, lastdata, lasttrenddata = parseData(data)
```

Listing 3.7 – Leitura dos dados do Arduino

O banco de dados possui 3 tabelas: uma tabela para armazenar o estado mais recente do sistema chamada Registers, outra tabela para armazenar o histórico das variáveis chamada TrendRegisters, e uma tabela para armazenar variáveis de controle (como o modo de operação do sistema por exemplo, e se o armazenamento histórico deve estar habilitado ou não) chamada OperationMode. Para armazenar dados no banco, primeiramente é necessário estabelecer uma conexão com o banco (ver Listing 3.8). A inserção do dado no banco consiste em acessar a tabela passando como parâmetro o dicionário criado anteriormente; por fim, deve-se executar o método save para concluir a operação (ver Listing 3.9).

```
#estabelece conexão com o banco de dados
if os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'WebSite.settings'):
  import django
  django.setup()
  from WebSite.operation.models import Registers, OperationMode
  from WebSite.trend.models import TrendRegister
  from django.utils import timezone
else:
  raise
  sys.exit(1)
```

Listing 3.8 – Conexão com o banco de dados

```
#envia dado para o banco de dados (lastdata contém últimos dados válidos)
reg = Registers(**lastdata)
reg.save()

#envia dado para o banco de dados
treg = TrendRegister(**lasttrenddata)
treg.save()
```

Listing 3.9 – Código necessário para inserção de dados no banco

[6] Como funciona um dicionário: (LUTZ, 2013)

As operações de leitura de banco de dados e escrita no banco são cíclicas, ou seja, executam a cada intervalo de tempo. O intervalo das operações é definido no início do código fonte. O tempo é determinado em milissegundos.

```
#intervalo de execução das operações
start_time = datetime.now()
readinterval = 100        #intervalo de leitura do arduino
sendDBinterval = 600      #intervalo de gravação na tabela Registers
sendTrendDBinterval = 300 #intervalo de gravação na tabela TrendRegisters
```

Listing 3.10 – Intervalo de execução das operações

3.3.3.2 Thread 2: Recebimento de Comandos do WebServer

A segunda thread implementa um servidor TCP que fica escutando na porta 8080. O servidor TCP implementado é assincrono, ou seja, não bloqueiam o processo em que estão sendo executados enquanto aguardam dados de algum cliente. Quando algum dado chega, é chamada uma função callback para processá-lo (PYTHON, 2016). Foi utilizado o pacote asyncore, que é disponibilizado na instalação padrão do Python para implementar o servidor assíncrono. A função callback chamada para interpretar os pacotes é a handle_read, mostrada no Listing 3.11

O WebServer envia comandos de duas formas. A primeira, envia apenas um byte, que é o código de comando a ser executado (ligar bomba, apagar histórico, entre outros); a função process_commands executa as ações necessárias de acordo com o comando recebido. A segunda forma, envia um pacote JSON[7], que contém o valor da velocidade da bomba a ser setada. O pacote JSON é processado, e o valor da velocidade é enviado para o Arduino na forma de um array de bytes.

3.3.4 WebServer

O código fonte completo do WebServer está disponibilizado no Github, no link: https://github.com/felipefonsecabh/PFC. Ao contrário dos códigos anteriores, o projeto do WebServer é composto de múltiplos arquivos.

A estrutura de um projeto django segue a estrutura mostrada na Figura 2.13. Um projeto é composto de várias aplicações (apps). Uma aplicação pode ser entendida como um módulo do programa, ou seja, um trecho que possui um conjunto de funcionalidades específicas e funciona de forma independente. A utilização de múltiplas apps no projeto é incentivada, pois permite o desenvolvimento de um sistema pouco acoplado.

A Figura 2.13 também mostra os arquivos que compõem uma aplicação. Destacam-se os aqueles que implementam o modelo MVC. Os arquivos de modelo implementam as

[7] O que é JSON: http://json.org/

```
    def handle_read(self):
      #recebe dado do cliente
      data = self.recv(50)
      if len(data) < 2:  #comandos digitais
        process_commands(data)
      else: #comando analogico
        try:
          #Exemplo de pacote JSON
          '{"pump_speed": 85.4}'

          ld = json.loads(data.decode('utf-8'))
          bytescommand = pack('f',ld['pump_speed'])
          bus.write_block_data(arduinoAddress,53,list(bytescommand))
        except Exception as err:
          print(str(err))
        finally:
          pass
```

Listing 3.11 – Função que interpreta comandos vindos do WebServer

estruturas de dados a serem utilizadas na aplicação. Cada estrutura de dados (classe) é mapeada na forma de uma tabela no banco de dados, e cada instância dessa estrutura na aplicação é uma linha na tabela criada, implementando assim o ORM, como mencionado na subseção 2.3.1.1.

Os arquivos urls e views implementam a parte do processamento de requisições do usuário e processo de apresentação de conteúdo. O arquivo url contém o mapeamento entre os recursos e as funções que devem ser executadas. Quando uma url é chamada, esse mapeamento é consultado e assim é chamada a função especificada.

As funções estão no arquivo views, que fazem o processamento propriamente dito. Basicamente, quando uma view é chamada, ela efetua operações com os modelos de dados quando necessário e retorna para o usuário algum arquivo de template.

Os templates são basicamente arquivos html que aceitam comandos disponibilizados pelo Framework para que se possa adicionar conteúdo dinâmico ao arquivo, antes de enviá-lo para o usuário. Alguns dos comandos mais utilizados são: exibir variáveis, cujos valores são preenchidos pelas views, “inserir templates dentro de outros”, o que possibilita reaproveitamento de código, comandos de decisão (if, else), de fluxo (for) entre outros. Os comandos possuem a sintaxe {% comando %}, exceto as variáveis, que são definidas por {{ variável }}. O fluxo de informações entre o usuário e uma aplicação django é exibido na Figura 3.5

A criação de um novo projeto django é feita através do comando startproject. O comando exibido no Listing 3.12 cria um projeto chamado WebSite. O projeto criado contém alguns arquivos pré-configurados como o arquivo wsgi.py, e o settings.py. O primeiro implementa o Web Server Gateway Interface (WSGI), uma especificação que permite

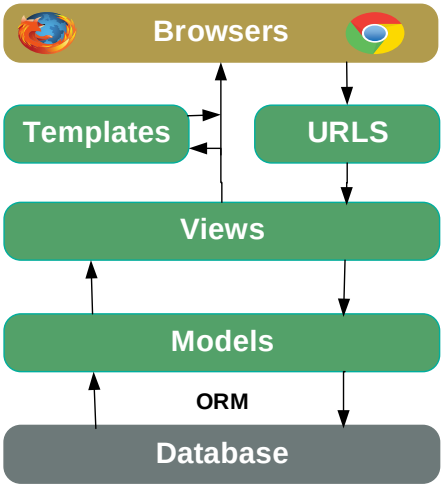

Figura 3.5 – Fluxo de Informações entre Usuário e Aplicação Django. Adaptado de: https://github.com/fga-gpp-mds/00-Disciplina/wiki/Padr%C3%B5es-Arquiteturais---MVC-X-Arquitetura-do-Django

que a aplicação desenvolvida possa ser executada em múltiplos servidores web (LAUBE, 2012); o segundo contém configurações gerais de um site, como por exemplo, qual será o banco de dados utilizado, caminho dos arquivos estáticos, etc.

```
# navegue até o diretóio destino da aplicação
$ django-admin startproject WebSite
```

Listing 3.12 – Função que interpreta comandos vindos do WebServer

O projeto é constituído de 3 aplicações: core, operation e trend. O funcionamento das mesmas será explicado a seguir:

3.3.4.1 Aplicação Core

É a aplicação base do sistema. Nesta aplicação são armazenadas todos os arquivos estáticos necessários, como arquivos de imagens, arquivos de estilo (css) e arquivos javascript que são utilizados pelas demais aplicações. Esta aplicação não implementa nenhum modelo de dados, pois sua função é somente servir de sustentação para outras aplicações.

O core possui dois templates: base.html e home.html. O template base consiste na estrutura geral da interface, que possui uma barra de navegação, um espaço central para inserção de conteúdo e um rodapé. Este template é utilizado pelos demais, que apenas preenchem os "espaços" disponibilizados no template base, para assim construir uma interface completa. A estrutura do arquivo base é mostrada no Listing 3.13

O template home.html implementa a página inicial do sistema, que é composta por duas imagens. O template home reutiliza o template base e preenche o espaço de

```
<1 - Inicialização da estrutura padrão de um arquivo html>
<2 - Inicialização de arquivos de estilo css comuns a todos os templates>
<3 - Espaço para arquivos de estilo adicionais>
<4 - Código da Barra de navegação>
<5 - Espaço Central para inserção de conteúdo por outros templates>
<6 - Código do rodapé>
<7 - Inicialização de arquivos javascript comuns a todos os templates>
<8 - Espaço para inserção de arquivos javascript adicionais>
```

Listing 3.13 – Template Base

conteúdo disponível. Não adiciona nenhum arquivo javascript ou css. O arquivo urls possui o mapeamento para apenas uma função no arquivo views, que é a chamada para a página inicial.

3.3.4.2 Aplicação Operation

A aplicação operation contém o maior número de funcionalidades do projeto. Nesta aplicação, são apresentadas para o usuário as informações sobre as variáveis que representam o estado do trocador de calor, bem como estão disponíveis os componentes que permitem a intervenção do usuário sobre o sistema. Ou seja, essa interface contém basicamente displays,para exibir informações, e botões para enviar comandos do usuário para o sistema.

Foram criados 3 modelos de dados para essa aplicação. Como mencionado na subseção 2.3.1.1, cada modelo de dados é mapeado em uma tabela no banco de dados. O modelo Display define uma estrutura de dados para exibição de dados analógicos. Através dessa definição, torna-se fácil alterações na unidade de engenharia, texto de descrição, entre outros fatores. O modelo Registers, é definido como o conjunto de informações enviadas pelo Arduino, ou seja, cada linha dessa tabela define o estado do trocador de calor em um dado instante. O modelo OperationMode consiste em armazenar informações auxiliares, como por exemplo se o armazenamento histórico está habilitado ou não. O mapeamento dessas estruturas no banco de dados está resumido na Tabela 3.7.

Tabela 3.7 – Modelos definidos no sistema

Displays	**Registers**		**OperationMode**
name: string	TimeStamp: DateTime	ColdFlow: float	OpMode: bool
UE: string	Temp1: float	PumpStatus: bool	TrendStarted: bool
description: string	Temp2: float	HeaterStatus: bool	
Tag: string	Temp3: float	ArduinoMode: bool	
Value: float	Temp4: float	EmergencyMode: bool	
	HotFlow: float	PumpSpeed: float	

Essa aplicação possui 5 funções definidas na view. A função main, que é acionada quando o usuário clica para entrar nessa tela. Enquanto o usuário se mantiver nessa tela, o navegador envia requisições constantes de atualização dos dados. Essa requisição de atualização chama a função refresh, que retorna para o cliente um pacote JSON com os dados mais atuais do sistema. Essa requisição é feita via AJAX[8], ou seja, não provoca a atualização de toda a página, apenas dos elementos que necessitam ser atualizados, o que otimiza a performance do sistema.

Quando o usuário clica em algum botão (desde que esteja habilitado), é chamada a view command, que repassa o comando para o gateway; quando o usuário movimenta o slider é chamada a view analogcommand, que repassa o comando de alteração de velocidade da bomba para o gateway. Por fim, a view export_csv, envia os dados históricos armazenados no banco para o usuário, quando solicitado.

A aplicação possui apenas um template, que reutiliza o template base e preenche o espaço de conteúdo. Para que o template funcione corretamente, foi necessário utilizar alguns arquivos css e javascript extras. A lista de arquivos e sua respectiva utilidade é descrita na Tabela 3.8.

Para que cada botão consiga enviar o respectivo comando para a view, criou-se um parâmetro em cada botão que recebe um índice de comando. Os números atribuídos são os mesmos definidos na Tabela 3.4. Os comandos definidos nessa tabela se referem apenas a comandos enviados para o Arduino. Existem ainda outros 3 comandos que são enviados para o gateway: habilitar / desabilitar o armazenamento de dados, e apagar os dados históricos do banco. Para esses botões foram atribuídos índices sequenciais a partir do último número presente na Tabela 3.4. Em todos os botões, os índices de comando são do tipo char.

Tabela 3.8 – Arquivos extras utilizados pela aplicação

Arquivo	**Função**
bootstrap-slider.css bootstrap-slider.js	Biblioteca mantida por Kemp e Kalkur (2017), que implementa o objeto slider da aplicação
raphael-2.1.4.js justgage.js	Biblioteca mantida por Djuricic (2016), que implementa o objeto gauge (barra de nível circular) da aplicação
bootstrap-confirmation.js	Biblioteca mantida por Sorel (2016) que implementa a confirmação após clique de botões
operation.js	Arquivo que executa a solicitação cíclica de dados para o servidor, bem como interpreta o retorno e atualiza a página para o usuário. Também monitora os eventos dos botões, e quando algum evento ocorre, envia os comandos para o servidor.

[8] Funcionamento do AJAX: https://www.w3schools.com/xml/ajax_intro.asp

3.3.4.3 Aplicação Trend

A aplicação trend basicamente exibe os dados analógicos em forma gráfica. Foram definidos 4 gráficos, para separar a visualização das grandezas. A relação entre as grandezas e em qual gráfico está exibida é mostrada Tabela 3.9.

Tabela 3.9 – Arquivos extras utilizados pela aplicação

Índice Área	**Grandezas**
1	Temperaturas do Ciclo de Água Quente
2	Temperaturas do Ciclo de Água Fria
3	Vazões de Água Quente e Fria
4	Velocidade da Bomba

Foi criado apenas 1 modelo para essa aplicação. O modelo TrendRegister é praticamente igual ao Register, porém não possui os campos digitais, apenas os analógicos.

A aplicação possui apenas 2 funcões definidas no arquivo view. A função main, é acionada quando o usuário clica para entrar na tela. Enquanto o usuário se mantiver nessa tela, o navegador envia requisições constantes de atualização de dados. Ou seja, nesse quesito funciona de forma idêntica à aplicação operation descrita anteriormente.

Para implementar a visualização em forma gráfica, foi necessário implementar bibliotecas de terceiros. Existem algumas bibliotecas open-source para exibir dados de forma gráfica na internet. A biblioteca escolhida foi a “SmoothieCharts”, disponibilizada por Walnes (2017).

O template dessa aplicação utiliza o template base e preenche o espaço central com os gráficos. Basicamente, o arquivo contém a definição das 4 áreas que serão exibidos os gráficos e adiciona dois arquivos javascript. O primeiro deles é a biblioteca mencionada anteriormente, e o segundo é o arquivo trend.js, que controla a frequência de requisição de dados para o servidor, bem como utiliza as funções disponibilizadas pela biblioteca para implementar a animação do gráfico.

4 Resultados

5 Conclusão

Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetuer nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Phasellus id magna. Duis malesuada interdum arcu. Integer metus. Morbi pulvinar pellentesque mi. Suspendisse sed est eu magna molestie egestas. Quisque mi lorem, pulvinar eget, egestas quis, luctus at, ante. Proin auctor vehicula purus. Fusce ac nisl aliquam ante hendrerit pellentesque. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi wisi. Etiam arcu mauris, facilisis sed, eleifend non, nonummy ut, pede. Cras ut lacus tempor metus mollis placerat. Vivamus eu tortor vel metus interdum malesuada.

Sed eleifend, eros sit amet faucibus elementum, urna sapien consectetuer mauris, quis egestas leo justo non risus. Morbi non felis ac libero vulputate fringilla. Mauris libero eros, lacinia non, sodales quis, dapibus porttitor, pede. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi dapibus mauris condimentum nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam sit amet erat. Nulla varius. Etiam tincidunt dui vitae turpis. Donec leo. Morbi vulputate convallis est. Integer aliquet. Pellentesque aliquet sodales urna.

Bibliografia

ARDUINO. **About Arduino**. 2017. Disponível em: <https://www.arduino.cc/>. Acesso em: 9 nov. 2017. Citado na página 10.

BAILEY, David; WRIGHT, Edwin. **Pratical SCADA for Industry**. 1. ed. [S.l.]: Newnes, 2003. ISBN 07506 58053. Citado na página 8.

BASKIYAR, S.; MEGHANNATHAN, N. A Survey of Contemporary Real-time Operating Systems. **Informatica**, v. 29, p. 233–240, 2005. Citado na página 2.

BRESEMAN, K. S. K. **A WEB DEVELOPER'S GUIDE TO COMMUNICATION PROTOCOLS (SPI, I2C, UART, GPIO)**. 2015. Disponível em: <https://tessel.io/blog/108840925797/a-web-developers-guide-to-communication-protocols>. Acesso em: 10 nov. 2017. Citado na página 12.

BREWPI. **About Brew Pi**. 2017. Disponível em: <https://www.brewpi.com/>. Acesso em: 24 out. 2017. Citado na página 13.

BURTON, Miles. **Dallas Temperature Library**. [S.l.: s.n.], 2016. Disponível em: <https://www.milesburton.com/Dallas_Temperature_Control_Library>. Citado na página 24.

DEVELOPERS, Mozilla. **About Django**. 2017. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction>. Acesso em: 13 nov. 2017. Citado na página 17.

DJURICIC, Bojan. **JustGage Library**. 2016. Disponível em: <https://github.com/toorshia/justgage>. Acesso em: 3 nov. 2017. Citado na página 34.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. Tese (Doutorado) – University of California, Irvine. Citado na página 2.

FILIPEFLOP. **Ultrasonic Library**. [S.l.: s.n.], 2015. Disponível em: <https://github.com/filipeflop/Ultrasonic>. Citado na página 24.

GORNI, Antônio Augusto. Introdução à prototipagem rápida e seus processos. **Plástico Industrial**, p. 230–239, mar. 2001. Disponível em: <http://www.gorni.eng.br/protrap.html>. Acesso em: 9 nov. 2017. Citado na página 10.

GREENFIELD, David. **Will SCADA Remain Relevant as Industry Advances?** [S.l.: s.n.], 7 set. 2017. Disponível em: <https://www.automationworld.com/will-scada-remain-relevant-industry-advances>. Acesso em: 11 out. 2017. Citado na página 2.

GREENFIELD, David. **How Embedded Systems Are Changing Automation**. 2013. Disponível em: <https://www.automationworld.com/article/technologies/embedded-control/how-embedded-systems-are-changing-automation>. Acesso em: 6 nov. 2017. Citado na página 1.

KEMP, Kyle; KALKUR, Rohit. **Bootstrap Slider Library**. 2017. Disponível em: <https://github.com/seiyria/bootstrap-slider>. Acesso em: 3 nov. 2017. Citado na página 34.

KIRUBASHANKAR, Ramesh Babau; K, Krishnamurthy; J, Indra. Remote monitoring system for distributed control of industrial plant process. **Journal of Scientific and Industrial Research**, v. 68, out. 2009. Citado na página 1.

KOWALCZYK, Kyle. **Enable I2C Interface on the Raspberry Pi**. 2017. Disponível em: <https://smallguysit.com/index.php/2017/10/28/python-benefits-using-virtual-environment/>. Acesso em: 30 out. 2017. Citado na página 27.

KREITH, Frank; MANGLIK, Raj M.; BOHN, Mark S. **Principles of Heat Transfer**. 7. ed. [S.l.]: Cengage Learning, 2011. p. 485–486. 784 p. Citado na página 1.

LALLI, AFelipe Micaroni; BUENO, Felippe Franco; ZACHARIAS, Guilherme Keese. **Evolução da Programação Web**. 2008. Faculdade Comunitária de Campinas Unidade III, Campinas. Citado na página 14.

LAUBE, Klaus Peter. **Entendendo o CGI, FastCGI e WSGI**. 2012. Disponível em: <https://klauslaube.com.br/2012/11/02/entendendo-o-cgi-fastcgi-e-wsgi.html>. Acesso em: 3 nov. 2017. Citado na página 32.

LEE, Tim Bernes. **Weaving the Web: The Original Design of the World Wide Web by its Inventor**. 1. ed. [S.l.]: HarperCollins, 2000. ISBN 0-06-251587-X. Citado na página 14.

LUTZ, Mark. **Learning Python**. 5. ed. [S.l.]: O'Reilly, 2013. p. 113–114. ISBN 978-1-449-35573-9. Citado na página 29.

MACEDO, Diego. **Protocolo HTTP: Estrutura, solicitações, respostas, métodos e códigos de status**. 2016. Disponível em: <http://www.diegomacedo.com.br/protocolo-http-estrutura-solicitacoes-respostas-metodos-e-codigos-de-status/>. Acesso em: 12 nov. 2017. Citado na página 16.

MALINOWSKI, A.; YU, H. Comparison of Embedded System Design for Industrial Applications. **IEEE Transactions on Industrial Informatics**, v. 7, n. 2, p. 244–254, maio 2011. ISSN 1551-3203. DOI: 10.1109/TII.2011.2124466. Citado na página 9.

MATT. **Enable I2C Interface on the Raspberry Pi**. 2014. Disponível em: <https://www.raspberrypi-spy.co.uk/2014/11/enabling-the-i2c-interface-on-the-raspberry-pi/>. Acesso em: 30 out. 2017. Citado na página 26.

NOVAZZI, Luis F. **Dinâmica e Controle de Redes de Trocadores de Calor**. 2007. Tese (Doutorado) – Universidade Estadual de Campinas, Faculdade de Engenharia Química, Campinas. Citado na página 1.

PEREIRA, Luis E. G.; ROCHA, Lucas Azevedo et al. **Projeto e implementação de sistema embarcado para monitoramento e controle do trocador de calor**. 2016. Universidade Federal de Minas Gerais, Belo Horizonte. Citado na página 6.

PEREIRA, Luiz Arthur Malta; CARVALHO, Francine Roberta et al. Software Embarcado, O Crescimento e as Novas Tendências Deste Mercado. **Revista de Ciências Exatas e Tecnologia**, v. 6, p. 85–94, 2011. Citado 2 vezes nas páginas 1, 2.

POLÔNIA, Pablo Valério. **Proposta de Arquitetura Orientada a Recursos para SCADA na Web**. Florianópolis: [s.n.], 2011. Citado 3 vezes nas páginas 2, 7, 8.

PRATYAKSA, Dipto. **How to Install Smbus I2C Module for Python3**. 2015. Disponível em: <http://www.linuxcircle.com/2015/05/03/how-to-install-smbus-i2c-module-for-python-3/>. Acesso em: 30 out. 2017. Citado na página 27.

PYTHON. **Asyncore - Asynchronous socket handler**. 2016. Disponível em: <https://docs.python.org/3.5/library/asyncore.html>. Acesso em: 31 out. 2017. Citado na página 30.

ROBOCORE. **Comparação entre Protocolos de Comunicação Serial**. Disponível em: <https://www.robocore.net/tutoriais/comparacao-entre-protocolos-de-comunicacao-serial.html>. Acesso em: 10 nov. 2017. Citado na página 12.

RODRIGUES, Lucas Alves; FARIA, Thiago Rodrigues. **Projeto de Automação para uma Estação de Tratamento de Múltiplas Fontes Alternativas de Água Utilizando Softwares e Hardwares Livres**. 2016. Instituto Federal de Educação, Ciência e Tecnologia Fluminense, Campos dos Goytacazes. Citado 2 vezes nas páginas 12, 13.

SOREL, Damien. **BootStrap Confirmation Library**. 2016. Disponível em: <http://bootstrap-confirmation.js.org/>. Acesso em: 3 nov. 2017. Citado na página 34.

STOFFREGEN, Paul. **OneWire Library**. [S.l.: s.n.], 2017. Disponível em: <https://www.pjrc.com/teensy/td_libs_OneWire.html>. Citado na página 24.

VALENZUELA, V. E. L. et al. Reusable hardware and software model for remote supervision of industrial Automation Systems using web technologies. In: 2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA). [S.l.: s.n.], set. 2013. p. 1–8. DOI: 10.1109/ETFA.2013.6647946. Citado na página 2.

WALNES, Joe. **Smoothie Chart Library**. 2017. Disponível em: <https://github.com/joewalnes/smoothie/releases>. Acesso em: 14 nov. 2017. Citado na página 35.

WANKE, Fabian; HUMPHREY, David. **Automation Meets Embedded Systems**. 2017. Disponível em: <https://www.automationworld.com/article/technologies/embedded-control/automation-meets-embedded-systems>. Acesso em: 6 nov. 2017. Citado na página 1.