

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-

53 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<sup>1</sup> e todos aqueles que contribuíram para que a produção de trabalhos acadêmicos conforme as normas ABNT com L<sup>A</sup>T<sub>E</sub>X fosse possível.

Agradecimentos especiais são direcionados ao Centro de Pesquisa em Arquitetura da Informação<sup>2</sup> da Universidade de Brasília (CPAI), ao grupo de usuários *latex-br*<sup>3</sup> e aos novos voluntários do grupo *abnT<sub>E</sub>X2*<sup>4</sup> que contribuíram e que ainda contribuirão para a evolução do abnT<sub>E</sub>X2.

---

<sup>1</sup> Os nomes dos integrantes do primeiro projeto abnT<sub>E</sub>X foram extraídos de <http://codigolivre.org.br/projects/abntex/>

<sup>2</sup> <http://www.cpai.unb.br/>

<sup>3</sup> <http://groups.google.com/group/latex-br>

<sup>4</sup> <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

Figura 2.1 –Trocador de Calor presente no LOP. . . . .	5
Figura 2.2 –Painel instalado na planta de trocador de calor . . . . .	6
Figura 3.1 –Arquitetura do sistema instalado . . . . .	7
Figura 3.2 –Arquitetura do sistema BrewPi . . . . .	10
Figura 3.3 –Nova Arquitetura Proposta para o Sistema . . . . .	10
Figura 3.4 –Arquitetura Detalhada . . . . .	13
Figura 3.5 –Estrutura do Código do Arduino . . . . .	14
Figura 4.1 –Estrutura de um pacote I2C . . . . .	21
Figura 4.2 –I2C habilitado . . . . .	23
Figura 4.3 –Fluxo de Informações entre Usuário e Aplicação Django . . . . .	28



# Lista de tabelas

Tabela 3.1 –Requisitos Funcionais do Sistema . . . . .	8
Tabela 3.2 –Comparação entre tecnologias . . . . .	9
Tabela 3.3 –Comparação entre versões do Raspberry PI . . . . .	11
Tabela 3.4 –Atribuições de cada Componente . . . . .	12
Tabela 4.1 –Definição das interpretações de comando . . . . .	22
Tabela 4.2 –Pacotes necessários para o projeto . . . . .	24
Tabela 4.3 –Modelos definidos no sistema . . . . .	30
Tabela 4.4 –Arquivos extras utilizados pela aplicação . . . . .	31



# 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

$\Gamma$	Letra grega Gama
$\Lambda$	Lambda
$\zeta$	Letra grega minúscula zeta
$\in$	Pertence





# Lista de Códigos

Figura 4.1 –Funções de Leitura dos sensores . . . . .	20
Figura 4.2 –Estrutura de dados do sistema . . . . .	22
Figura 4.3 –Comandos para criação de um ambiente virtual . . . . .	24
Figura 4.4 –Comandos para criação de um ambiente virtual . . . . .	24
Figura 4.5 –Comando para a instalação de pacotes Python . . . . .	24
Figura 4.6 –Inicialização da comunicação I2C . . . . .	25
Figura 4.7 –Leitura dos dados do Arduino . . . . .	25
Figura 4.8 –Conexão com o banco de dados . . . . .	26
Figura 4.9 –Código necessário para inserção de dados no banco . . . . .	26
Figura 4.10 –Intervalo de execução das operações . . . . .	26
Figura 4.11 –Função que interpreta comandos vindos do WebServer . . . . .	27
Figura 4.12 –Função que interpreta comandos vindos do WebServer . . . . .	28
Figura 4.13 –Template Base . . . . .	29



# Sumário

<b>Lista de Códigos</b>	<b>23</b>
<b>1 Introdução</b>	<b>1</b>
<i>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</i>	
1.1 Motivação e Justificativa	1
1.2 Objetivos	2
1.3 Local de Realização	3
1.4 Estrutura da Monografia	3
<b>2 Revisão Bibliográfica</b>	<b>5</b>
<i>Teste, teste</i>	
2.1 Laboratório de Operações e Processos	5
2.2 Controle e Monitoramento de Processos Industriais	6
<b>3 Metodologia</b>	<b>7</b>
<i>Este capítulo apresenta a arquitetura do sistema implementado. Primeiramente, são descritos os requisitos funcionais do sistema, que orientam a montagem da estrutura. São definidos os componentes do sistema e suas interconexões, sendo que esta definição é acompanhada de um breve descritivo do funcionamento.</i>	
3.1 Estado Atual	7
3.2 Requisitos Funcionais do Sistema	8
3.3 Critérios para Escolha do Sistema	9
3.4 Arquitetura do Sistema	10
3.4.1 Escolha da Versão do Raspberry PI	11
3.4.2 Arquitetura Detalhada do Sistema	12
3.4.2.1 Funcionamento do Arduino	13
3.4.2.2 Comunicação Entre Arduino e Gateway	15
3.4.2.3 Funcionamento do Gateway	15
3.4.2.4 Comunicação entre WebServer e Gateway	16
3.4.2.5 WebServer	16
<b>4 Implementação</b>	<b>19</b>
<i>Este capítulo contém explicações sobre trechos não triviais dos códigos referentes aos componentes projetados. Além disso contém informações sobre o procedi-</i>	

*mento de preparação do Raspberry PI para ser integrado ao sistema.*

4.1	Código do Arduino . . . . .	19
4.1.1	Leitura dos Sensores . . . . .	19
4.1.2	Comunicação I2C . . . . .	20
4.2	Preparação do Raspberry PI . . . . .	21
4.2.1	Instalação do sistema operacional . . . . .	21
4.2.2	Ativação do protocolo I2C . . . . .	22
4.2.3	Instalação de Pacotes - Python . . . . .	23
4.3	Código do Gateway . . . . .	24
4.3.1	Thread 1: Leitura de Dados do Arduino e Escrita no Banco . . . . .	24
4.3.2	Thread 2: Recebimento de Comandos do WebServer . . . . .	26
4.4	WebServer . . . . .	27
4.4.1	Aplicação Core . . . . .	29
4.4.2	Aplicação Operation . . . . .	29
<b>5</b>	<b>Resultados . . . . .</b>	<b>33</b>
<b>6</b>	<b>Conclusão . . . . .</b>	<b>35</b>
	<b>Bibliografia . . . . .</b>	<b>37</b>
	<b>Apêndices . . . . .</b>	<b>41</b>
	<b>APÊNDICE A Quisque libero justo . . . . .</b>	<b>43</b>
	<b>APÊNDICE B Nullam elementum urna vel imperdiet sodales elit ipsum pharetra ligula ac pretium ante justo a nulla curabitur tristique arcu eu metus . . . . .</b>	<b>45</b>
	<b>Anexos . . . . .</b>	<b>47</b>
	<b>ANEXO A Morbi ultrices rutrum lorem. . . . .</b>	<b>49</b>
	<b>ANEXO B Cras non urna sed feugiat cum sociis natoque penatibus et magnis disparturient montes nascetur ridiculus mus . . . . .</b>	<b>51</b>
	<b>ANEXO C Fusce facilisis lacinia dui . . . . .</b>	<b>53</b>

# 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

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;
- 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 perturbações 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 mais amigável de operação, que contribui para uma atuação mais rápida e assertiva;
- Melhor acessibilidade aos dados coletados, o que permite a equipe de engenharia e de gestão analisar e 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 <sup>1</sup>, bem como é possível verificar a utilização de sistemas embarcados.<sup>2</sup> 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 estão cada vez mais poderosos e oferecem cada vez mais recursos (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 protocolo proprietário.

Colocar nesse parágrafo informações sobre a evolução na utilização de sistemas Web no controle de processo, abordando a necessidade crescente de comunicação de vários sistemas.

Este trabalho apresenta o desenvolvimento de um sistema de baixo custo para monitoramento e operação via Web de um trocador de calor 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 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:

<sup>1</sup> <https://www.wonderware.com/hmi-scada/what-is-scada/>

<sup>2</sup> [http://files.comunidades.net/mutcom/ARTIGO\\_SIST\\_EMB.pdf](http://files.comunidades.net/mutcom/ARTIGO_SIST_EMB.pdf)

- 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 pesquisa.

## 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 e Processos

O Laboratório de Operações e Processos (LOP) 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 <sup>1</sup>.

O LOP possui quatro plantas didáticas: **explicar aqui brevemente as 4 plantas constituintes do laboratório**

O trocador de calor existente no laboratório, cuja imagem pode ser vista na [Figura 2.1](#), é do tipo **tubular**.



Figura 2.1 – Trocador de Calor presente no LOP.

Este trocador de calor foi instalado no laboratório em **colocar aqui a história do trocador de calor** e originalmente possuía apenas a indicação das medidas de temperatura. A imagem destaca os sensores de temperatura instalados inicialmente. A medição de vazão era inferida através da utilização de uma Calha Parshall<sup>2</sup>.

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

<sup>1</sup> <http://www.deq.ufmg.br/departamento/infraestrutura>

<sup>2</sup> <http://www.dec.ufcg.edu.br/saneamento/PARSHALL.html>

para monitoramento, operação 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<sup>3</sup> 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.

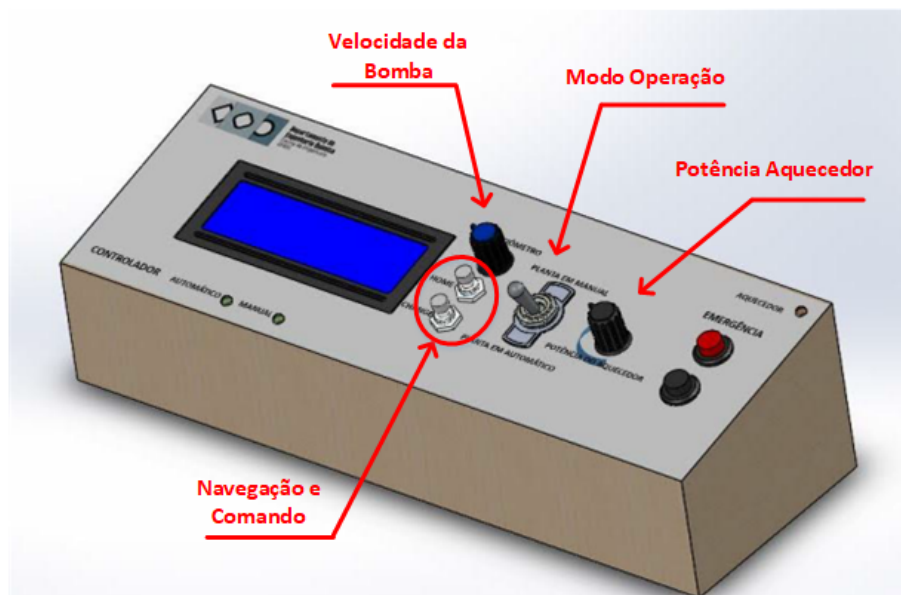


Figura 2.2 – Painel instalado na planta de trocador de calor.  
Adaptado de (PEREIRA; ROCHA et al., 2016)

## 2.2 Controle e Monitoramento de Processos Industriais

<sup>3</sup> <https://store.arduino.cc/usa/arduino-due>

### 3 Metodologia

*Este capítulo apresenta a arquitetura do sistema implementado. Primeiramente, são descritos os requisitos funcionais do sistema, que orientam a montagem da estrutura. São definidos os componentes do sistema e suas interconexões, sendo que esta definição é acompanhada de um breve descritivo do funcionamento.*

#### 3.1 Estado Atual

A Arquitetura do sistema atual instalado na planta de trocador de calor está representada na [Figura 3.1](#). Pelo painel é possível efetuar a leitura das variáveis de processo, cuja leitura é feita pelo Arduino, bem como comandar a bomba e o aquecedor. É importante observar que o controle da potência do aquecedor não passa pelo Arduino, e que o sistema não recebe nenhum feedback sobre a velocidade da bomba, bem como sobre a potência do aquecedor. Apesar de estar previsto no projeto de [Pereira, Rocha et al. \(2016\)](#), nenhum controle em malha fechada está operacional.

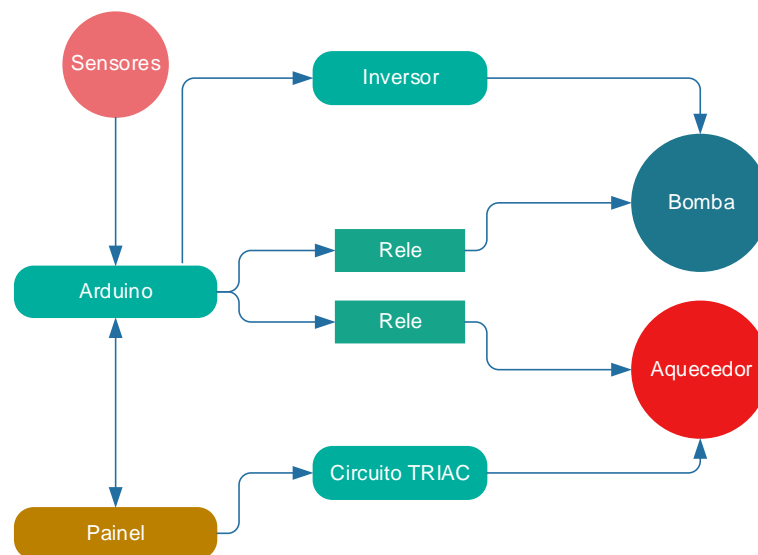


Figura 3.1 – Arquitetura do sistema instalado

O intuito do projeto é implementar um sistema de monitoramento e controle remoto, sem que haja alteração infraestrutura já montada. Em resumo, o objetivo é implementar um sistema que necessite apenas se comunicar com o Arduino para executar as suas tarefas.

## 3.2 Requisitos Funcionais do Sistema

Para determinar a arquitetura do sistema a ser implementado, foram levantados os requisitos funcionais do sistema, ou seja, quais informações devem ser transmitidas ao operador, bem como quais as ações que este pode realizar no sistema. 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

Após levantamento dos requisitos, é possível concluir que as arquiteturas citadas na [seção 2.2](#) podem ser utilizadas. É possível utilizar um PC que executa um software SCADA tradicional como utilizado por [Gonçalves, Silva e Batista \(2015\)](#). Basicamente, o Arduino foi programado de forma que ele consiga utilizar um protocolo industrial comumente utilizado por ferramentas SCADA, no caso o protocolo Modbus<sup>1</sup> foi utilizado. Também é possível utilizar uma arquitetura totalmente embarcada como utilizada no sistema *BrewPi* ([BREWPI, 2017](#)).

<sup>1</sup> <http://www.ni.com/white-paper/52134/pt/>

### 3.3 Critérios para Escolha do Sistema

Para possibilitar a comparação entre a arquitetura SCADA tradicional e embarcada e consequentemente ajudar na tomada de decisão, algumas características pontuais do sistema foram avaliadas. Existem características subjetivas e objetivas, sendo que as características subjetivas foram avaliadas de acordo com a experiência na utilização das ferramentas pelo desenvolvedor. Como existem diversos softwares SCADA no mercado, um software específico foi escolhido para comparação. O E3, que é fornecido pela Elipse Software<sup>2</sup>, foi escolhido devido ao grau de conhecimento do desenvolvedor sobre a ferramenta. Os critérios e avaliações estão apresentados na Tabela 3.2.

Tabela 3.2 – Comparação entre tecnologias

Arquitetura Critério	Tecnologia Embarcada	
	SCADA (E3)	
Custo	Alto	Baixo
Complexidade do desenvolvimento das interfaces gráficas	Baixa	Alta
Complexidade da implementação da comunicação com Arduino	Baixa	Média
Facilidade para interoperar com outros sistemas	Média	Alta
Multiplataforma	Não	Sim
Interface se adapta à resolução do dispositivo	Não	Sim
Suporte a vários clientes conectados simultaneamente	Sim	Sim
Consumo de energia	Médio	Baixo

A grande disparidade entre esses sistemas basicamente é o custo em a relação a demanda de mão-de-obra especializada para desenvolvimento de sistemas embarcados. Conhecimentos de ferramentas SCADAs já estão difundidos, facilitando o desenvolvimento de aplicações utilizando essas ferramentas. Contudo, esses softwares são caros, com exceção de algumas ferramentas *opensource* como por exemplo o SCADABR<sup>3</sup>.

Desde o início da modernização do laboratório, o custo era uma das restrições mais fortes, o que favoreceu a opção pela tecnologia embarcada. Além disso, a oportunidade de utilizar tecnologias emergentes no monitoramento de processos contribuiu ainda mais para a utilização de sistemas embarcados.

<sup>2</sup> <https://www.elipse.com.br/produto/elipse-e3/>

<sup>3</sup> <http://www.scadabr.com.br/>

### 3.4 Arquitetura do Sistema

A arquitetura pensada para o sistema foi inspirada no projeto BrewPi. Esse sistema foi concebido para controlar a temperatura de um recipiente com líquido, bem como para permitir o monitoramento via qualquer dispositivo que possua um navegador Web. A Arquitetura deste sistema é mostrada na [Figura 3.2](#). Basicamente, o Arduino, é encarregado de executar a malha de controle e enviar as informações para um RaspberryPi, que implementa um WebServer. Os usuários que desejam visualizar e/ou modificar as informações, devem abrir um navegador e se conectar ao WebServer para obter os dados.

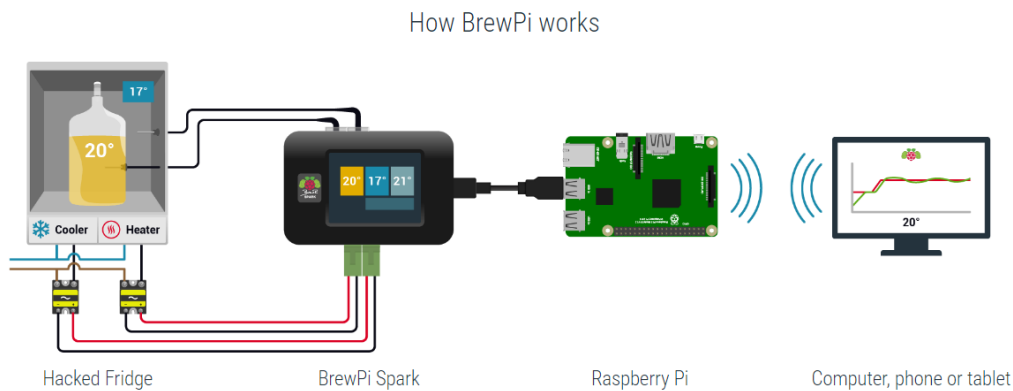


Figura 3.2 – Arquitetura do sistema BrewPi. Fonte: <https://www.brewpi.com/>

Para o trocador de calor, a proposta é semelhante. A adição de um RaspberryPi no sistema atual é simples de ser realizada e está em conformidade com a condição de não modificar as instalações atuais. Dessa forma, a arquitetura proposta simplificada para o sistema com o Raspberry é exibida na [Figura 3.3](#). A Arquitetura detalhada do sistema, como por exemplo, os softwares a serem desenvolvidos em cada dispositivo, bem como a forma de interação entre eles serão tópicos abordados posteriormente.

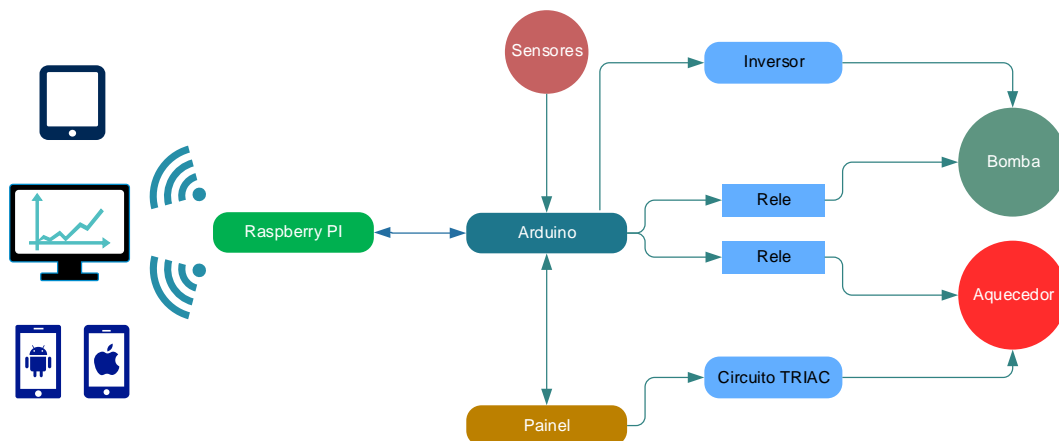


Figura 3.3 – Nova Arquitetura Proposta para o Sistema

É importante salientar que existem outros dispositivos similares ao Raspberry PI como o BeagleBone Black<sup>4</sup> e a recente e também poderosa placa DragonBoard 410c<sup>5</sup>. Qualquer uma das placas poderia ser utilizada no projeto. A escolha do Raspberry PI foi influenciada pela extensa base de conhecimento e documentação já produzida pelos usuários, aliado ao menor custo da placa. As demais placas por serem mais novas, não dispõem de muitos documentos e exemplos, o que aumenta o tempo de desenvolvimento de um novo projeto.

Ainda em relação as placas utilizadas, verifica-se que existem vários modelos distintos de Raspberry PI. Portanto é necessário definir qual a versão é a mais adequada ao projeto.

### 3.4.1 Escolha da Versão do Raspberry PI

O Raspberry PI possui algumas versões, que se diferem em tamanho, capacidade de memória, processamento e componentes. Um resumo das características das principais versões do Raspberry é exibido na [Tabela 3.3](#)

Tabela 3.3 – Comparação entre versões do Raspberry.  
Adaptado de [https://en.wikipedia.org/wiki/Raspberry\\_Pi](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

A última versão do Raspberry apresenta uma grande vantagem por possuir Wifi nativamente. Isso evita a utilização de adaptadores para utilizar a comunicação Wireless no dispositivo. Além disso, possui maior poder de processamento, conta com um chip

<sup>4</sup> <https://beagleboard.org/black>

<sup>5</sup> <https://developer.qualcomm.com/hardware/dragonboard-410c>

mais moderno, e suporta uma capacidade maior de corrente em estado de sobrecarga. Portanto, o Raspberry 3 foi escolhido. Na próxima seção detalhados os sistemas a serem executados dentro do Raspberry e como estes se relacionam entre si e com o Arduino.

### 3.4.2 Arquitetura Detalhada do Sistema

A [Figura 3.1](#) exibe a visão geral do sistema, ou seja, não exibe como é a comunicação entre os componentes. Como dito anteriormente, não faz parte do escopo do projeto promover alterações nas instalações atuais, portanto, é necessário focar apenas em como o Raspberry será programado e como será realizada a sua comunicação com o Arduino.

Os componentes foram projetados de forma que cada um funcione da forma mais independente possível. Dessa forma, o impacto no sistema caso haja alguma alteração de componente, em termos de esforço de desenvolvimento, é minimizado.

A arquitetura detalhada do sistema é exibida na [Figura 3.4](#). Foram definidos 3 componentes: o programa que é executado no Arduino; o Gateway e o WebServer, **que são** programas que são executados no Raspberry. O termo componente deve ser entendido nesse contexto como um programa, por isso o banco de dados não deve ser interpretado como um componente. Optou-se por criar dois programas no Raspberry PI, de forma que as funcionalidades do sistema fossem corretamente agrupadas. As atribuições de cada programa estão resumidas na [Tabela 3.4](#)

Tabela 3.4 – Atribuições de cada Componente

	Atribuições
Arduino	1 - Interface com sensores e Atuadores
	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 as atribuições 4 e 5 envolvem uma comunicação entre Arduino e Gateway, e as atribuições 7 e 9 envolvem uma comunicação entre o WebServer e Gateway. Portanto, é necessário definir protocolos de comunicação para ambos os casos.



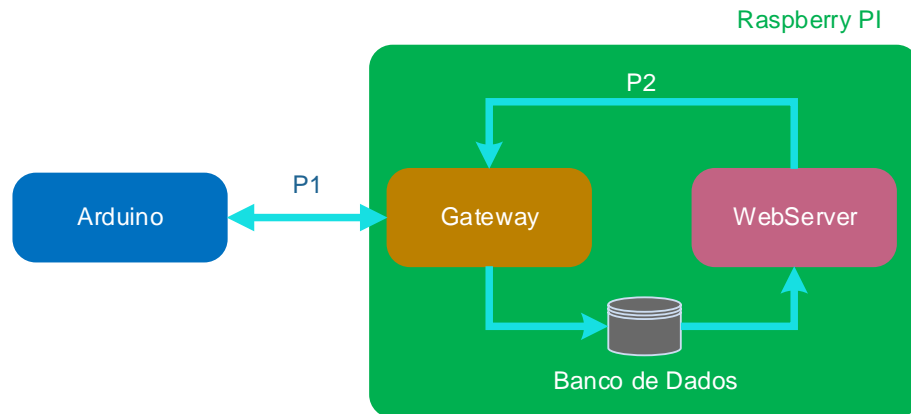


Figura 3.4 – Arquitetura Detalhada

Já foi dito que os componentes foram projetados de forma a possibilitar a substituição de um componente sem alterar os demais. Porém, alterações no protocolo de comunicação necessariamente levam a modificações nos componentes envolvidos. Em resumo, os protocolos de comunicação devem ser mantidos.

#### 3.4.2.1 Funcionamento do Arduino

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.

A estrutura do código, que está exibida na [Figura 3.5](#), 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 3.4](#). As trocas de mensagens com o Gateway são feitas por interrupção, ou seja, estão fora da função *loop* do Arduino.

O programa criado seguiu o paradigma de programação procedural, ou seja, o programa foi bastante modularizado. Esse tipo de prática acelera a identificação de uma funcionalidade (atribuição) no código fonte, o que facilita a manutenção deste. Outro benefício é a possibilidade de expansão de funcionalidades, como por exemplo a implementação de malhas de controle no código. **Para isso, um Estado Automático cujas atribuições fossem executar as malhas de controle poderia ser adicionado ao sistema. Para que este estado entre em funcionamento, bastaria adicionar mais uma condição de teste no código, ou seja, uma alteração bastante simples.**

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.

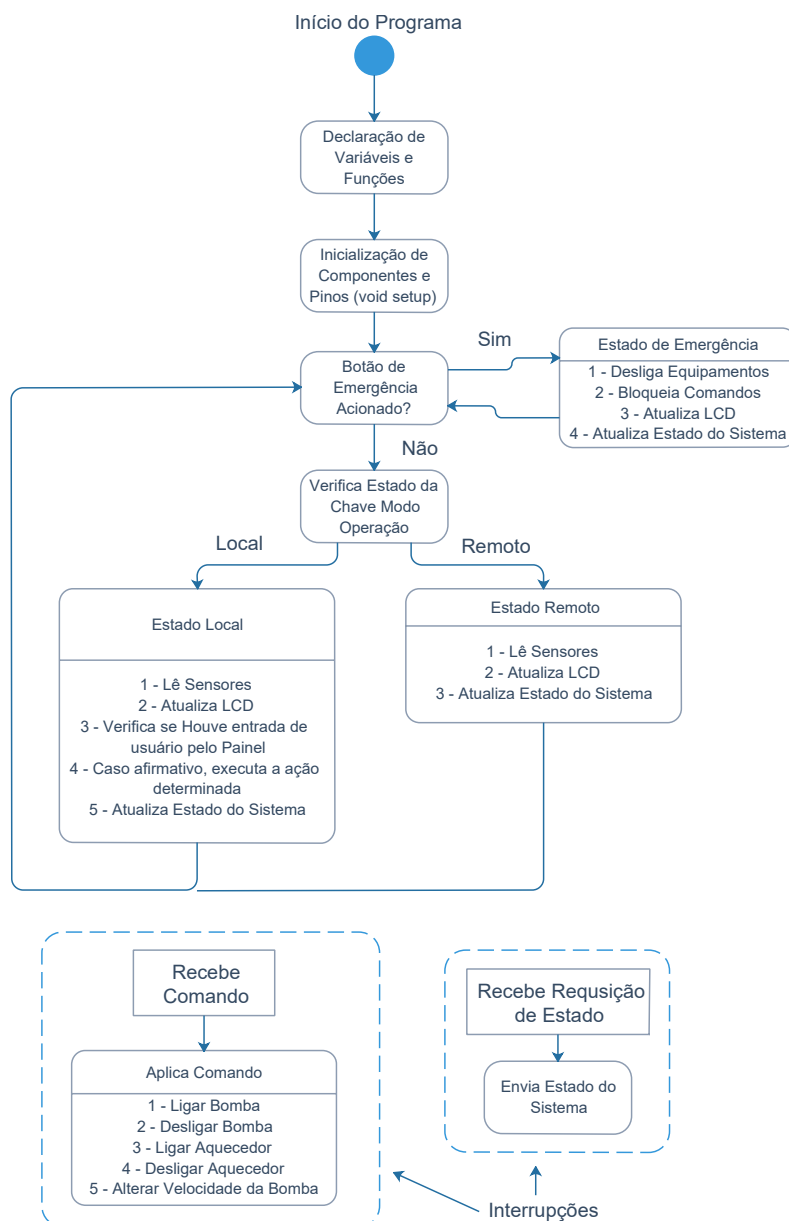


Figura 3.5 – Estrutura do Código do Arduino

### 3.4.2.2 Comunicação Entre Arduino e Gateway

Ao observar as atribuições 3 e 4 da [Tabela 3.4](#), verifica-se que a comunicação entre Arduino e Gateway é do tipo mestre-escravo, ou seja, o dispositivo mestre (Gateway) sempre inicia a comunicação, e o escravo (Arduino) responde quando solicitado. Existem diversos protocolos que implementam esse tipo de comunicação.

É necessário escolher um protocolo que funcione em meio físico serial, uma vez que nenhum *Shield* Ethernet será utilizado. É possível portanto utilizar algum protocolo industrial, disponibilizado na forma de bibliotecas, ou então algum dos protocolos seriais descritos na seção [colocar aqui a seção da rev sobre protocolos](#), ou seja: UART, SPI, I2C.

O protocolo industrial aberto encontrado com maior frequência em outras referências e implementações é o Modbus. É possível baixar pelo menos 5 bibliotecas que implementam o protocolo. A utilização de Modbus apresenta a vantagem de facilitar a integração com diferentes sistemas, uma vez que é um protocolo amplamente difundido e utilizado. Contudo, ao adicionar a biblioteca Modbus no Arduino, o código aumenta consideravelmente o tamanho, pois as bibliotecas implementam praticamente todo o protocolo, o que é desnecessário para o projeto atual, uma vez que a quantidade de informações trafegadas entre os componentes é pequena.

O protocolo SPI possui uma taxa de transferência máxima maior que I2C e UART, porém a implementação da troca das mensagens entre os dispositivos é a mais complexa. O protocolo SPI exige o recebimento de informação enquanto está enviando, o que é ruim em para o propósito desse sistema, pois torna-se necessário o tratamento de informações indesejadas. A comunicação UART permite a implementação mais simples de troca de mensagens. Porém, é a mais lenta das 3, além de ser assíncrona. Além disso, a troca de mensagens é através de strings, portanto, o tamanho dos frames envolvidos na troca de mensagens é maior.

O protocolo I2C se apresenta como um meio termo entre SPI e UART, em termos de complexidade das trocas de mensagens e velocidade de comunicação. Além disso, é um protocolo síncrono, e a velocidade típica de utilização, que é 100kbps, atende plenamente ao projeto. Portanto, o I2C será utilizado no projeto. Existe uma biblioteca nativa que implementa o I2C para o Arduino, bem como existem bibliotecas I2C para diversas linguagens de programação (necessário para o Gateway). Os detalhes dos frames de comunicação serão explorados no capítulo de [implementação](#).

### 3.4.2.3 Funcionamento do Gateway

O programa do Gateway foi desenvolvido em Python, embora possa ser escrito também em Java e C/C++, que são linguagens também suportadas pelo Raspberry. O Python foi escolhido por ser uma linguagem de programação de alto nível, ou seja,

uma linguagem que está próxima da linguística humana. Essa característica facilita o entendimento de programas, e acelera o processo de aprendizado da mesma.

O processo do Gateway foi dividido em duas Threads. Uma thread é responsável por solicitar os dados do arduino e enviar ao banco de dados, e a outra responsável por aguardar comandos vindos do WebServer, e repassar ao Arduino. A divisão se mostra uma decisão acertada. Enquanto a tarefa de leitura dos dados e envio ao banco é executada em um intervalo fixo, o recebimento dos dados do WebServer é essencialmente assíncrono, pois os usuários podem enviar comandos em qualquer momento. Portanto, implementar ambas as funcionalidades em uma única Thread seria inviável. Além disso, essa implementação permite o repasse do comando enviado pelo WebServer para o Arduino tão logo a mensagem seja recebida. As Threads compartilham apenas um recurso, que é o canal I2C, para escrita e leitura dos dados para o Arduino.

Vale a pena colocar uma imagem para ilustrar o que foi dito no texto?

#### 3.4.2.4 Comunicação entre WebServer e Gateway

A comunicação entre WebServer e Gateway é feita via TCP/IP. Uma das threads do Gateway implementa um servidor TCP assíncrono que fica “escutando” em uma determinada porta. A cada comando de usuário, o WebServer abre um cliente, se conecta ao servidor TCP, envia a informação necessária e posteriormente fecha a conexão. Dessa forma, evita-se manter conexões estabelecidas de forma desnecessária. A linguagem Python possui pacotes que implementam servidores e clientes TCP, ou seja, o desenvolvedor não necessita se preocupar com detalhes de implementação do protocolo. Sendo assim, o foco está apenas no conteúdo das mensagens envolvidas.

As mensagens trafegadas consistem em um conjunto strings. Para simplificar o processo de decodificação da mensagem, bem como facilitar a captura de erros no transporte, decidiu-se transmitir as informações em pacotes JSON. Como dito na seção [colocar aqui a seção que fala sobre json](#), uma das vantagens de se utilizar JSON é a fácil interpretação das mensagens, tanto pelos humanos, quanto pelos softwares.

#### 3.4.2.5 WebServer

Conforme foi abordado na seção [colocar a seção sobre frameworks web](#), existem frameworks web para diversas linguagens de programação, inclusive existem muitos frameworks para uma mesma linguagem de programação. A linguagem Python foi mantida para o desenvolvimento do WebServer, pelos mesmos motivos mencionados na seção anterior.

[Primakovski \(2017\)](#) e [Brown \(2015\)](#) abordam sobre as diferenças entre os diversos frameworks web existentes para a linguagem Python. O framework escolhido foi o Django

devido a 4 motivos:

1. É um dos frameworks mais utilizados. Portanto, conta com uma vasta e completa documentação.
2. A complexidade de uma aplicação escrita em Django é praticamente uniforme. Ou seja, uma aplicação de pequeno porte ou grande porte, resultam em um código de complexidade próxima.
3. O framework Django é adequado para usuários que querem desenvolver aplicações web rapidamente, sem fazer muitas decisões sobre a infraestrutura da aplicação. Ou seja, o Django já fornece uma infraestrutura básica para possibilitar desenvolvimento imediato
4. Consultas a banco de dados não precisam ser feitas utilizando código SQL. O Django provê uma estrutura de forma que é possível manipular dados do banco através de códigos Python. Isso facilita, por exemplo, algumas funcionalidades do Gateway.

Para apresentação da interface para o cliente são utilizadas as linguagens HTML e CSS. A parte da animação e interatividade da página é desenvolvida utilizando JavaScript. Essas 3 linguagens são utilizadas por quase todos os sistemas web existentes atualmente.



## 4 Implementação

*Este capítulo contém explicações sobre trechos não triviais dos códigos referentes aos componentes projetados. Além disso contém informações sobre o procedimento de preparação do Raspberry PI para ser integrado ao sistema.*

### 4.1 Código do Arduino

O código do Arduino foi elaborado utilizando o diagrama descrito na [Figura 3.5](#). O código fonte completo está disponibilizado no Github<sup>1</sup>, no link <https://github.com/felipefonsecabh/ArduinoCode/blob/ArduinoNoNavigation/ArduinoCode.ino>

#### 4.1.1 Leitura dos Sensores

Para a leitura dos sensores foram utilizadas bibliotecas desenvolvidas e mantidas por terceiros. Essas bibliotecas são de código aberto, ou seja, outros usuários podem contribuir para o desenvolvimento e melhoria do código. Geralmente, os projetos das bibliotecas são disponibilizados no GitHub. É importante verificar a versão das bibliotecas utilizadas, pois uma versão pode funcionar de forma diferente da outra.

Para a leitura dos sensores de temperatura foi utilizada a biblioteca Dallas Temperature, mantida por [Burton \(2016\)](#). A versão utilizada no projeto é a mais atual (versão 3.7.6). A Dallas Temperature possui uma dependência de uma outra biblioteca. Dessa forma, é necessário utilizar também a biblioteca OneWire, atualmente mantida por [Stof-fregen \(2017\)](#). A versão utilizada desta biblioteca também é a mais atual (2.3.3).

Para a leitura da vazão de água quente, foi utilizada a biblioteca Ultrasonic criada por [FilipeFlop \(2015\)](#). Aparentemente, essa biblioteca não está sendo mantida por ninguém. Existem outras bibliotecas para o sensor ultrassônico HC-SR404 disponíveis na internet. O funcionamento do sensor é descrito em [Thomsen \(2011\)](#).

Para a leitura de vazão de água fria não foi utilizada nenhuma biblioteca. O sensor de vazão consiste em um dispositivo que envia pulsos ao Arduino. Quanto mais pulsos enviados, maior é a vazão. Para a medição da vazão, 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 contagem de pulsos se dá por interrupção.

Como dito na [subseção 3.4.2.1](#), as funções que fazem a leitura dos sensores e convertem os valores para unidade de engenharia foram as mesmas utilizadas por [Pereira, Rocha et al. \(2016\)](#). Não faz parte do escopo desse projeto calibrar os sensores novamente.

O código correspondente a leitura dos valores analógicos é mostrado no Listing 4.1.

<sup>1</sup> <https://www.oficinadanet.com.br/post/14791-o-que-github>

```

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 4.1 – Funções de Leitura dos sensores

### 4.1.2 Comunicação I2C

O Arduino disponibiliza na sua instalação padrão, a biblioteca Wire, que permite o Arduino se comunicar via I2C com outros dispositivos. Neste projeto o Arduino atua como slave da comunicação, ou seja, não inicia a comunicação. Portanto, ele responde a algum dispositivo mestre apenas quando solicitado.

Quando o mestre envia um comando de escrita, atua-se uma interrupção, que 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 referentes ao sistema foram colocados em uma estrutura de dados.



São 7 variáveis do tipo float (4 temperaturas, 2 vazões e ainda o valor da velocidade da bomba, e) e 4 informações digitais (estado da bomba e do aquecedor, estado do botão de emergência, estado da chave local/remoto), que foram agrupadas em um byte. A [Figura 4.1](#) mostra a estrutura de um pacote I2C enviado pelo mestre. Basicamente, o I2C trafega dados em um array de bytes. O primeiro byte é um número que indica o índice de um comando. Esse valor é arbitrário e é definido pelo desenvolvedor que implementa o protocolo. O segundo byte indica a quantidade de bytes dos dados transmitidos. Esse valor é útil para verificar a consistência do pacote recebido. O restante do frame consiste nos dados propriamente ditos.

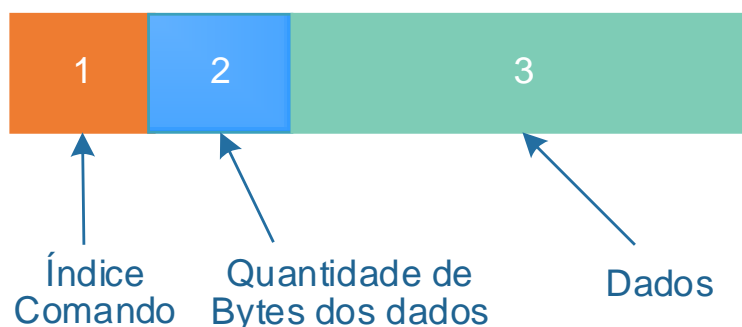


Figura 4.1 – Estrutura de um pacote I2C

Como dito anteriormente, os dados que representam o estado do sistema, não correspondem a um array de bytes. Portanto, foi necessário converter as informações float e digitais para bytes para possibilitar a transmissão do mesmo. No Arduino isso foi feito utilizando a declaração union. Essa técnica permite variáveis de diferentes tipos ocupem uma mesma região de memória, o que possibilita que a estrutura de dados anteriormente criada possua a sua representação em um array de bytes, o que é necessário para transmissão via I2C. O [Listing 4.2](#) corresponde a implementação feita.

Para correta interpretação dos comandos enviados 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 4.1](#). 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`.

## 4.2 Preparação do Raspberry PI

### 4.2.1 Instalação do sistema operacional

Conforme mencionado na [seção X](#), o Raspberry não possui um HD interno. Portanto, é necessário instalar o sistema operacional em um microUSB, que faz o papel de HD. O Rpi suporta vários sistemas operacionais, sendo que o sistema operacional padrão

```

//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 checksum;
};

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

```

Listing 4.2 – Estrutura de dados do sistema

Tabela 4.1 – Definição das interpretações de comando

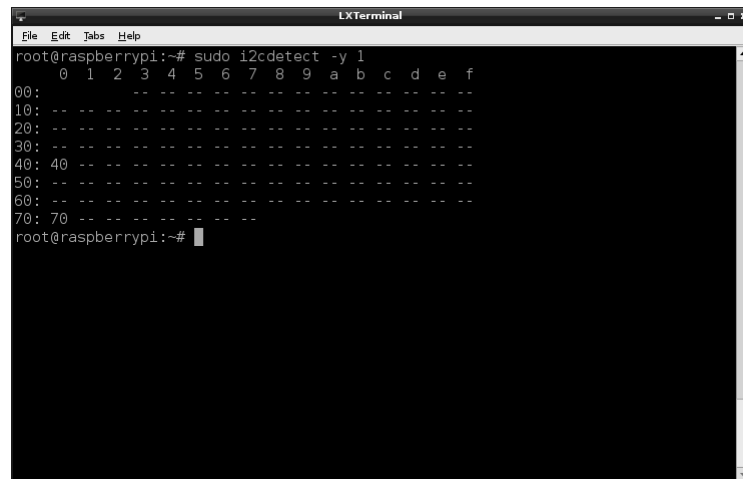
Índice Comando	Ação
6	Enviar dados do sistema para o mestre
49	Ligar a bomba
50	Desligar a bomba
51	Ligar aquecedor
52	Desligar o aquecedor
53	Alterar velocidade da bomba

é o Raspian, uma adaptação de uma distribuição Debian<sup>2</sup>. Foi utilizada a versão “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.

### 4.2.2 Ativação do protocolo I2C

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). Para verificar se o i2c está corretamente habilitado, deve-se abrir um terminal e digitar o comando “sudo i2cdetect -y 1”. O resultado é exibido na Figura 4.2.

<sup>2</sup> <https://www.debian.org/intro/about>



```
LXTerminal
File Edit Tabs Help
root@raspberrypi:~# sudo i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
30:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- --
50:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
60:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- -- -- -- -- -- -- -- --
root@raspberrypi:~#
```

Figura 4.2 – I2C habilitado

### 4.2.3 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 virtuais 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 4.3](#). Uma vez instalado, é necessário ativar o ambiente virtual. O [Listing 4.4](#) contém o código para ativá-lo. Após ativado, devem-se instalar os pacotes necessários para o projeto. Os pacotes e as versões instaladas estão listados na [Tabela 4.2](#)

Todos os pacotes, com exceção do Smbus, podem ser instalados através do comando exibido no [Listing 4.5](#). 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@raspberrypi $ cd pfc_env

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

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

Listing 4.3 – Comandos para criação de um ambiente virtual

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

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

Listing 4.4 – Comandos para criação de um ambiente virtual

Tabela 4.2 – 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@raspberrypi $ pip install Django==1.9.6
```

Listing 4.5 – Comando para a instalação de pacotes Python

## 4.3 Código do Gateway

O código fonte completo do gateway está disponibilizado no Github, no link: <https://github.com/felipefonsecabh/PFC/blob/PyServeri2c/arduinosever.py>. Conforme mencionado na subseção 3.4.2.3, o programa consiste em 2 threads que são executadas continuamente, ou seja em loop infinito.

### 4.3.1 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 4.6 – 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 índice do comando (foi definido na [Tabela 4.1](#) que o índice do comando da solicitação de dados era 6), 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`<sup>3</sup>. 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 apenas um valor para verificação da integridade do pacote. Após a separação em variáveis, os dados foram organizados em um dicionário para facilitar o envio para o banco. ([LUTZ, 2013](#)).

```
#faz requisicao pelos dados  
block = bus.read_i2c_block_data(arduinoAddress,6,30)  
#conversao do array de bytes em variaveis  
data = unpack('7f2b',bytes(block))  
#inserção dos dados em um dicionário  
bstatus, lastdata, lasttrenddata = parseData(data)
```

Listing 4.7 – 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 4.8](#)). A inserção do dado 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 4.9](#)).

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.

<sup>3</sup> <https://docs.python.org/2/library/struct.html>

```
#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 4.8 – 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 4.9 – Código necessário para inserção de dados no banco

```
#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 4.10 – Intervalo de execução das operações

### 4.3.2 Thread 2: Recebimento de Comandos do WebServer

A segunda thread implementa um servidor TCP que fica escutando na porta 8080. Servidores assíncronos 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 4.11](#)

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, 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.

```
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 4.11 – Função que interpreta comandos vindos do WebServer

## 4.4 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 **FiguraX**. 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 **FiguraX** mostra o conjunto de arquivos que compõem uma aplicação. Destacam-se os arquivos que implementam o modelo MVC. Os arquivos de modelo implementam as 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. Esse procedimento é um exemplo da utilização do ORM<sup>4</sup> do django.

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

<sup>4</sup> <https://www.fullstackpython.com/object-relational-mappers-orms.html>

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 4.3

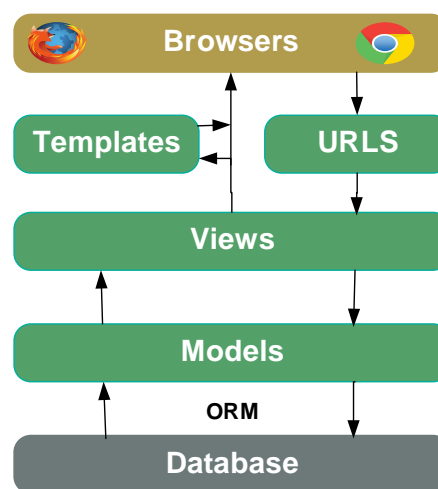


Figura 4.3 – Fluxo de Informações entre Usuário e Aplicação Django. Adaptado de: <https://github.com/fga-gpp-mds/00-Disciplina/wiki/Padr%C3%B5es-Arquiteturais---MVC-X-Arquitetura-do-Django>

A criação de um novo projeto django é feita através do comando startproject. O comando exibido no Listing 4.12 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 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 diretório destino da aplicação
$ django-admin startproject WebSite
```

Listing 4.12 – 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:



### 4.4.1 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 4.13](#)

```
<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 4.13 – Template Base

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 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.

### 4.4.2 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 [seção](#), 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 4.3](#).

Tabela 4.3 – Modelos definidos no sistema

Displays	Registers	OperationMode
name: string	TimeStamp: DateTime	OpMode: bool
UE: string	Temp1: float	TrendStarted: bool
description: string	Temp2: float	
Tag: string	Temp3: float	
Value: float	Temp4: float	
	HotFlow: float	
	ColdFlow: float	
	PumpStatus: bool	
	HeaterStatus: bool	
	ArduinoMode: bool	
	EmergencyMode: bool	
	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, 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 a biblioteca 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 4.4](#). Para que cada botão consiga enviar o respectivo comando para a view, criou-se um parâmetro que recebe um índice de comando. Os números atribuídos são os mesmos definidos na [Tabela 4.1](#). A diferença é que no arquivo de template a representação do comando está em char, enquanto na tabela consta a representação do

comando em decimal. Isso ocorre porque o Arduino interpreta o comando, que foi enviado na forma de string, através da representação decimal. A relação entre um caractere e sua representação numérica pode ser consultada na tabela ASCII<sup>5</sup>.

Tabela 4.4 – Arquivos extras utilizados pela aplicação

Arquivo	Função
bootstrap-slider.css bootstrap-slider.js	Biblioteca mantida por <a href="#">Kemp e Kalkur (2017)</a> , que implementa o objeto slider da aplicação
raphael-2.1.4.js justgage.js	Biblioteca mantida por <a href="#">Djuricic (2016)</a> , que implementa o objeto gauge (barra de nível circular) da aplicação
bootstrap-confirmation.js	Biblioteca mantida por <a href="#">Sorel (2016)</a> 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.

<sup>5</sup> <https://en.wikipedia.org/wiki/ASCII>



## 5 Resultados



## 6 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, consectetur 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 consectetur 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

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

BREWPI. **About Brew Pi**. 2017. Disponível em: <<https://www.brewpi.com/>>. Acesso em: 24 out. 2017. Citado na página 8.

BROWN, Ryan. **Django vs Flask vs Pyramid: Choosing a Python Web Framework**. 2015. Disponível em: <<https://www.airpair.com/python/posts/django-flask-pyramid>>. Acesso em: 27 out. 2017. Citado na página 16.

BURTON, Miles. **Dallas Temperature Library**. [S.l.: s.n.], 2016. Disponível em: <[https://www.milesburton.com/Dallas\\_Temperature\\_Control\\_Library](https://www.milesburton.com/Dallas_Temperature_Control_Library)>. Citado na página 19.

DJURICIC, Bojan. **JustGage Library**. 2016. Disponível em: <<https://github.com/toorshia/justgage>>. Acesso em: 3 nov. 2017. Citado na página 31.

FILIPEFLOP. **Ultrasonic Library**. [S.l.: s.n.], 2015. Disponível em: <<https://github.com/filipeflop/Ultrasonic>>. Citado na página 19.

GONÇALVES, Alexandre José; SILVA, Jair Rodrigues da; BATISTA, João Carlos. **Sistema Didático de Automação Baseado em Computador para Seleção de Esferas**. 2015. Universidade Tecnológica Federal do Paraná, Curitiba. Citado na página 8.

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.

KEMP, Kyle; KALKUR, Rohit. **Bootstrap Slider Library**. 2017. Disponível em: <<https://github.com/seiyria/bootstrap-slider>>. Acesso em: 3 nov. 2017. Citado na página 31.

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 23.

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.

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 28.

LUTZ, Mark. **Learning Python**. 5. ed. [S.l.]: O'Reilly, 2013. p. 113–114. ISBN 978-1-449-35573-9. Citado na página 25.

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 22.

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 3 vezes nas páginas 5–7, 19.

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 na página 2.

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 23.

PRIMAKOVSKI, Bogdan. **17 Best Python Web Frameworks to Learn in 2017**. 27 maio 2017. Disponível em: <<http://steelkiwi.com/blog/best-python-web-frameworks-to-learn/>>. Acesso em: 27 out. 2017. Citado na página 16.

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 26.

SOREL, Damien. **Bootstrap Confirmation Library**. 2016. Disponível em: <<http://bootstrap-confirmation.js.org/>>. Acesso em: 3 nov. 2017. Citado na página 31.

STOFFREGEN, Paul. **OneWire Library**. [S.l.: s.n.], 2017. Disponível em: <[https://www.pjrc.com/teensy/td\\_libs\\_OneWire.html](https://www.pjrc.com/teensy/td_libs_OneWire.html)>. Citado na página 19.

THOMSEN, Adilson. **Como conectar o Sensor Ultrassônico HC-SR04 ao Arduino**. 2011. Disponível em: <<https://www.filipeflop.com/blog/sensor-ultrassonico-hc-sr04-ao-arduino/>>. Acesso em: 28 out. 2017. Citado na página 19.



## Apêndices



## APÊNDICE A – Quisque libero justo

Quisque facilisis auctor sapien. Pellentesque gravida hendrerit lectus. Mauris rutrum sodales sapien. Fusce hendrerit sem vel lorem. Integer pellentesque massa vel augue. Integer elit tortor, feugiat quis, sagittis et, ornare non, lacus. Vestibulum posuere pellentesque eros. Quisque venenatis ipsum dictum nulla. Aliquam quis quam non metus eleifend interdum. Nam eget sapien ac mauris malesuada adipiscing. Etiam eleifend neque sed quam. Nulla facilisi. Proin a ligula. Sed id dui eu nibh egestas tincidunt. Suspendisse arcu.





# APÊNDICE B – Nullam elementum urna vel imperdiet sodales elit ipsum pharetra ligula ac pretium ante justo a nulla curabitur tristique arcu eu metus

Nunc velit. Nullam elit sapien, eleifend eu, commodo nec, semper sit amet, elit. Nulla lectus risus, condimentum ut, laoreet eget, viverra nec, odio. Proin lobortis. Curabitur dictum arcu vel wisi. Cras id nulla venenatis tortor congue ultrices. Pellentesque eget pede. Sed eleifend sagittis elit. Nam sed tellus sit amet lectus ullamcorper tristique. Mauris enim sem, tristique eu, accumsan at, scelerisque vulputate, neque. Quisque lacus. Donec et ipsum sit amet elit nonummy aliquet. Sed viverra nisl at sem. Nam diam. Mauris ut dolor. Curabitur ornare tortor cursus velit.

Morbi tincidunt posuere arcu. Cras venenatis est vitae dolor. Vivamus scelerisque semper mi. Donec ipsum arcu, consequat scelerisque, viverra id, dictum at, metus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut pede sem, tempus ut, porttitor bibendum, molestie eu, elit. Suspendisse potenti. Sed id lectus sit amet purus faucibus vehicula. Praesent sed sem non dui pharetra interdum. Nam viverra ultrices magna.

Aenean laoreet aliquam orci. Nunc interdum elementum urna. Quisque erat. Nullam tempor neque. Maecenas velit nibh, scelerisque a, consequat ut, viverra in, enim. Duis magna. Donec odio neque, tristique et, tincidunt eu, rhoncus ac, nunc. Mauris malesuada malesuada elit. Etiam lacus mauris, pretium vel, blandit in, ultricies id, libero. Phasellus bibendum erat ut diam. In congue imperdiet lectus.



# Anexos



## ANEXO A – Morbi ultrices rutrum lorem.

Sed mattis, erat sit amet gravida malesuada, elit augue egestas diam, tempus scelerisque nunc nisl vitae libero. Sed consequat feugiat massa. Nunc porta, eros in eleifend varius, erat leo rutrum dui, non convallis lectus orci ut nibh. Sed lorem massa, nonummy quis, egestas id, condimentum at, nisl. Maecenas at nibh. Aliquam et augue at nunc pellentesque ullamcorper. Duis nisl nibh, laoreet suscipit, convallis ut, rutrum id, enim. Phasellus odio. Nulla nulla elit, molestie non, scelerisque at, vestibulum eu, nulla. Ut odio nisl, facilisis id, mollis et, scelerisque nec, enim. Aenean sem leo, pellentesque sit amet, scelerisque sit amet, vehicula pellentesque, sapien.



# ANEXO B – Cras non urna sed feugiat cum sociis natoque penatibus et magnis disparturient montes nascetur ridiculus mus

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, consectetur 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.





## ANEXO C – Fusce facilisis lacinia dui

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.