



**UNIVERSIDADE FEDERAL DO CEARÁ  
CENTRO DE TECNOLOGIA  
DEPARTAMENTO DE ENGENHARIA ELÉTRICA  
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

**RÔMULO BANDEIRA PIMENTEL DRUMOND**

**SISTEMA DE MONITORAMENTO EM TEMPO REAL  
DE CONSUMO DE ÁGUA VIA WEB**

**FORTALEZA  
2017**

RÔMULO BANDEIRA PIMENTEL DRUMOND

SISTEMA DE MONITORAMENTO EM TEMPO REAL  
DE CONSUMO DE ÁGUA VIA WEB

Monografia apresentada ao Curso de Engenharia Elétrica do Departamento de Engenharia Elétrica da Universidade Federal do Ceará, como requisito parcial para obtenção do Título de Graduado em Engenharia Elétrica.

Orientador: Prof. Dr.-Ing. Sérgio Daher.

FORTALEZA

2017

Dados Internacionais de Catalogação na Publicação  
Universidade Federal do Ceará  
Biblioteca Universitária

Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

---

D858s Drumond, Rômulo Bandeira Pimentel.

Sistema de monitoramento em tempo real de consumo de água via web / Rômulo Bandeira Pimentel  
Drumond. – 2017.  
67 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia,  
Curso de Engenharia Elétrica, Fortaleza, 2017.  
Orientação: Prof. Dr. Sergio Daher.

1. Monitoramento em tempo real. 2. Monitoramento de consumo de água. 3. Internet das Coisas. 4.  
IoT. 5. Flask. I. Título.

CDD 621.3

---

RÔMULO BANDEIRA PIMENTEL DRUMOND

TRABALHO DE CONCLUSÃO DE CURSO: SISTEMA DE MONITORAMENTO EM  
TEMPO REAL DE CONSUMO DE ÁGUA VIA WEB

Esta monografia foi julgada adequada para a obtenção do grau de Graduado em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Aprovada em: \_\_\_\_ / \_\_\_\_ / \_\_\_\_.

BANCA EXAMINADORA

---

Prof. Dr.-Ing. Sergio Daher (Orientador)

Universidade Federal do Ceará (UFC)

---

Prof. Dr. Fabrício Gonzalez Nogueira

Universidade Federal do Ceará (UFC)

---

Prof. Dr. Bruno Ricardo de Almeida

Universidade de Fortaleza (Unifor)

Ao meu pai, Edilson Pimentel Drumond.

## **AGRADECIMENTOS**

Ao Prof. Dr.-Ing. Sergio Daher, pela sua prestatividade em me ajudar e sagacidade na resolução de problemas que surgiram durante este trabalho.

À todos os meus amigos da universidade. Todos aqueles que me ajudaram e me fizeram rir nessa longa graduação em Engenharia Elétrica. Não me atrevo a listar todos os nomes, me tomariam muitas páginas assim como muitas lembranças.

À Zizi, por ser a namorada alegre que amenizou meus momentos de estresse.

À minha família por me apoiar no desenvolvimento deste trabalho e na construção do meu caráter.

## RESUMO

Este trabalho tem como objetivo a construção de uma solução completa de Internet das Coisas (IoT) que permita monitorar e analisar o consumo de água. O projeto inclui o desenvolvimento de um hidrômetro inteligente conectado a internet e um aplicativo *web* com acesso a um banco de dados, permitindo mediante a um *website* ver gráficos de vazão em tempo real e histórico de consumo. O hidrômetro inteligente foi fabricado utilizando um sensor de vazão acessível da empresa SAIER, a plataforma *open-source* Arduino UNO e o módulo Wi-Fi de baixo custo ESP8266. O *firmware* desenvolvido neste trabalho foi voltado ao Arduino, que se comunicou com o módulo Wi-Fi por comandos AT através da UART, e foi escrito na linguagem de programação própria da IDE do Arduino. O lado do servidor da aplicação *web* foi programado em *Flask*, uma *framework* de *Python*, utilizando-se de extensões para agilizar o processo de desenvolvimento e do lado do cliente foram escritos *scripts* em *Javascript* para adicionar interatividade a página *web*. O protótipo desenvolvido durante o trabalho foi instalado numa torneira residencial e foi possível analisar o consumo durante cinco dias, visualizar o perfil de consumo de um usuário escovando os dentes e ser notificado ao final de cada dia sobre o estado do consumo. Após o período de testes, foi possível afirmar que o sistema funcionara conforme o esperado, informações sobre o consumo foram enviadas em tempo real ao servidor e através da interface *web* foi possível constatar que os dados de consumo condiziam com a realidade assim como foram apresentados de uma forma mais intuitiva e amigável. Pode-se afirmar que durante este trabalho um sistema eficaz de monitoramento de consumo de água foi desenvolvido e é capaz de ser utilizado em diversos contextos para adquirir informações sobre o uso de água em tempo real.

**Palavras-chave:** Monitoramento em tempo real. Monitoramento de consumo de água. Internet das Coisas. IoT. ESP8266. Wi-FI. *Flask*.

## ABSTRACT

This work aims the construction of a complete Internet of Things (IoT) solution that allows monitoring and analyzing water consumption. The project includes the development of an internet-connected intelligent hydrometer and a web application with database access, allowing through a website to view real-time flow graphs and consumption history. The intelligent hydrometer was fabricated using a non expensive flow sensor from SAIER, an open-source Arduino UNO platform and a low-cost Wi-Fi module ESP8266. The firmware developed in this work was directed to Arduino, which communicated with the Wi-Fi module by AT commands through the UART, and was written in the programming language of Arduino IDE. The server-side of the web application was programmed in Flask, a Python framework, using extensions to streamline the development process and the client-side scripts were coded in Javascript to add interactivity to the web page. The prototype developed during this academic work was installed in a residential faucet and it was possible to analyze the consumption during five days, to visualize the consumption profile of a user brushing their teeth and to be notified at the end of each day about the state of consumption. After the test period, it was possible to assert that the system worked as expected, information about the consumption was sent in real time to the server and through the web interface it was possible to verify that the consumption data were consistent with reality as they were presented in a more intuitive and friendly way. During this work an effective water consumption monitoring system has been developed and can be used in several contexts to acquire information on water usage in real time.

**Keywords:** Real-time monitoring. Monitoring of water consumption. Internet of Things. IoT. ESP8266. Wi-Fi. Flask.

## LISTA DE FIGURAS

Figura 1 - Esquema completo do sistema desenvolvido.....	15
Figura 2 - Representação gráfica da internet, com uma possível conexão entre dois dispositivos destacada em verde.....	16
Figura 3 - Protocolo HTTP representado graficamente com seus pares de requisição e resposta.....	17
Figura 4 - Representação gráfica do WebSocket.....	18
Figura 5 - Conectando os dispositivos à internet quando todos possuem capacidade IP.....	19
Figura 6 - Conectando os dispositivos à internet quando apenas um possui capacidade IP ..	19
Figura 7 - Sensor de fluxo utilizado.....	21
Figura 8 - Visão interna de um sensor de vazão similar.....	21
Figura 9 - Teste proposto para calibração do sensor de vazão.....	23
Figura 10 - Arduino UNO.....	25
Figura 11 - ESP8266-01.....	25
Figura 12 - Módulo adaptador do ESP8266-01.....	27
Figura 13 - Circuito final do hidrômetro inteligente.....	28
Figura 14 - Algoritmo do firmware.....	29
Figura 15 - Esquema do banco de dados.....	32
Figura 16 - Rede de hidrômetros inteligentes.....	36
Figura 17 - Página inicial do site.....	37
Figura 18 - Seção da página web intitulada "Meta".....	37
Figura 19 - Notificação na página web da ultrapassagem de 80% da meta diária.....	38
Figura 20 - Gráfico da meta quando essa é ultrapassada.....	38
Figura 21 - Seção que apresenta a vazão de água em tempo real.....	39
Figura 22 - Seção que mostra o histórico de consumo de água.....	40
Figura 23 - Seção intitulada "Medidas para reduzir desperdícios".....	40
Figura 24 - E-mail notificando o consumo de 80% da meta diária.....	41
Figura 25 - E-mail do relatório diário.....	41
Figura 26 - Protótipo instalado.....	42
Figura 27 - Visão interna do protótipo construído.....	42
Figura 28 - Consumo de água em relação a meta diária e mensal.....	43
Figura 29 - Vazão em tempo real para um usuário escovando os dentes.....	43
Figura 30 - Histórico de consumo de água pela torneira visto pelo site.....	44

Figura 31 - E-mail com o relatório diário de consumo do último dia.....45

## **LISTA DE TABELAS**

Tabela 1 - Algumas especificações técnicas do sensor de vazão.....	22
Tabela 2 - Resultados dos testes para a calibração do sensor de vazão.....	23
Tabela 3 - Especificações técnicas do microcontrolador ATmega328P.....	24
Tabela 4 - Especificações técnicas do ESP8266.....	26
Tabela 5 - Exemplo de comandos AT disponíveis para o ESP8266.....	26
Tabela 6 - Especificações técnicas do módulo adaptador.....	27
Tabela 7 - Especificações técnicas do diodo 1N5819.....	28
Tabela 8 - Histórico de consumo da torneira em litros.....	44

## LISTA DE ABREVIATURAS E SIGLAS

6LoWPAN	<i>IPv6 over Low power Personal Area Networks</i>
ABNT	Associação Brasileira de Normas Técnicas
AMI	<i>Advanced Metering Infrastructure</i>
BLE	<i>Bluetooth Low Energy</i>
HTML	<i>HyperText Markup Language</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IoE	<i>Internet of Everything</i>
IoT	<i>Internet of Things</i>
IP	<i>Internet Protocol</i>
ISP	<i>Internet Service Provider</i>
LAN	<i>Local Area Networks</i>
SI	<i>Système International d'unités</i>
URL	<i>Uniform Resource Locator</i>
WSN	<i>Wireless Sensor Network</i>

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO.....</b>	<b>14</b>
<b>1.1</b>	<b>Metodologia.....</b>	<b>15</b>
<b>1.2</b>	<b>Estrutura do trabalho.....</b>	<b>15</b>
<b>2</b>	<b>REVISÃO BIBLIOGRÁFICA.....</b>	<b>16</b>
<b>2.1</b>	<b>A Internet e a <i>Web</i>.....</b>	<b>16</b>
<b>2.2</b>	<b>Aplicações <i>Web</i>.....</b>	<b>18</b>
<b>2.3</b>	<b>Conectividade no panorama da Internet das Coisas.....</b>	<b>18</b>
<b>3</b>	<b>PROJETO DO MEDIDOR INTELIGENTE E <i>FIRMWARE</i>.....</b>	<b>21</b>
<b>3.1</b>	<b>Sensor de vazão de água.....</b>	<b>21</b>
<b>3.2</b>	<b>Arduino UNO.....</b>	<b>24</b>
<b>3.3</b>	<b>Módulo Wi-Fi.....</b>	<b>25</b>
<b>3.4</b>	<b><i>Hardware</i> do Medidor Inteligente.....</b>	<b>27</b>
<b>3.5</b>	<b>Algoritmo do <i>firmware</i>.....</b>	<b>29</b>
<b>4</b>	<b>APLICAÇÃO WEB.....</b>	<b>31</b>
<b>4.1</b>	<b>Design do banco de dados.....</b>	<b>31</b>
<b>4.1.1</b>	<b><i>Tabela</i> “<i>Usuários</i>”.....</b>	<b>32</b>
<b>4.1.2</b>	<b><i>Tabela</i> “<i>Cargos</i>”.....</b>	<b>33</b>
<b>4.1.3</b>	<b><i>Tabela</i> “<i>Medidores</i>”.....</b>	<b>33</b>
<b>4.1.4</b>	<b><i>Tabela</i> “<i>Modelos de Medidores</i>”.....</b>	<b>34</b>
<b>4.1.5</b>	<b><i>Tabela</i> “<i>Metas</i>”.....</b>	<b>34</b>
<b>4.1.6</b>	<b><i>Tabela</i> “<i>Medições</i>”.....</b>	<b>35</b>
<b>4.1.7</b>	<b><i>Tabela</i> “<i>Hierarquia</i>”.....</b>	<b>35</b>
<b>4.1.8</b>	<b><i>Tabela</i> “<i>Genealogia</i>”.....</b>	<b>35</b>
<b>4.2</b>	<b>O site.....</b>	<b>36</b>
<b>4.2.1</b>	<b><i>Meta</i>.....</b>	<b>37</b>
<b>4.2.2</b>	<b><i>Vazão de água</i>.....</b>	<b>39</b>
<b>4.2.3</b>	<b><i>Histórico de consumo de água</i>.....</b>	<b>39</b>
<b>4.2.4</b>	<b><i>Medidas para reduzir desperdícios</i>.....</b>	<b>40</b>
<b>4.3</b>	<b>Outros recursos.....</b>	<b>41</b>
<b>5</b>	<b>RESULTADOS EXPERIMENTAIS.....</b>	<b>42</b>
<b>6</b>	<b>CONCLUSÃO.....</b>	<b>46</b>
	<b>REFERÊNCIAS.....</b>	<b>48</b>

<b>APÊNDICE A – FIRMWARE DO MEDIDOR INTELIGENTE.....</b>	<b>50</b>
<b>APÊNDICE B – CÓDIGO DO LADO DO SERVIDOR.....</b>	<b>52</b>

## 1 INTRODUÇÃO

A região Nordeste do Brasil, segundo Suassuna (2002), apresenta baixo volume pluviométrico anual e alta frequência de secas. Das várias medidas tomadas pelo Governo para reduzir os efeitos da falta de água a conscientização da população através de racionamentos, com taxas de contenção para aqueles que não cumprirem uma meta mensal de consumo, é uma das mais perceptíveis ao cidadão comum. Nesse contexto, consumidores de pequeno e grande porte enfrentam dificuldades ao tentarem reduzir a conta de água:

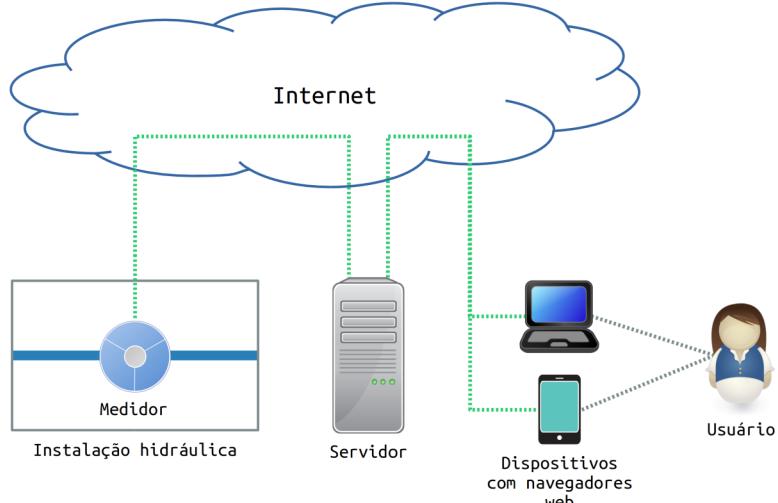
- a) como saber se novas ações e medidas estão surtindo efeito se a conta de água é entregue mensalmente;
- b) que parte da instalação hidráulica gasta a maior fatia da conta de água;
- c) como identificar e localizar vazamentos antes que eles já tenham resultado em grandes desperdícios de água e dinheiro.

Uma solução para esses problemas baseia-se no monitoramento remoto, ou telemetria, do consumo de água. Através da telemetria um indivíduo seria capaz de acompanhar em tempo real o consumo da instalação, possuindo uma aptidão maior em reduzir o consumo de água. O uso de vários dispositivos de monitoramento remoto ao longo da instalação permite, se seccionando-a, determinar a participação de cada parte da instalação na conta de água e, se cobrindo pontos de um longo encanamento, localizar mais facilmente a origem de vazamentos.

Para implantar a solução proposta um dispositivo capaz de medir o consumo de água e enviá-lo pela internet foi desenvolvido. Tal aparato se enquadra num novo setor da tecnologia: Internet das Coisas, ou em inglês *Internet of Things* (IoT), também conhecida como Internet de Tudo, *Internet of Everything* (IoE). O termo IoT possui várias definições na literatura, conforme IEEE (2015), sendo que todas englobam a ação de adicionar à uma “coisa” (sendo ela uma televisão, carro, lâmpada ou um medidor de consumo de água) sensores, poder de processamento e conectividade à internet. Dessa forma é gerada uma conexão entre o mundo físico e virtual, facilitando o fluxo de informação e automação de processos e sistemas.

Para uma maior robustez as informações adquiridas pelo medidor em vez de enviadas diretamente ao usuário são recebidas por um servidor que armazena e trata as informações para apresentá-las numa interface *web*, ou simplesmente um *site* da *Web*. A Figura 1 exibe um esquema completo do sistema.

Figura 1 - Esquema completo do sistema desenvolvido



Fonte: Elaborada pelo autor.

## 1.1 Metodologia

Consiste inicialmente da construção de um medidor inteligente, composto por um sensor de vazão de água conectado a um microcontrolador ATmega328P que solicita a um módulo Wi-Fi enviar informações ao servidor. Após isso a aplicação *web* é desenvolvida para receber, guardar, processar e apresentar essas medições de forma coerente ao usuário. No final um teste é realizado instalando o dispositivo numa torneira residencial.

## 1.2 Estrutura do trabalho

Este trabalho foi estruturado da seguinte maneira:

- a) Capítulo 2, uma revisão bibliográfica com conceitos relacionados a internet e conectividade de dispositivos;
- b) Capítulo 3, o projeto do *hardware* do medidor inteligente é apresentado esmiuçando cada componente utilizado e o algoritmo do *firmware*;
- c) Capítulo 4, a aplicação *web* é vista em detalhes junto com a estrutura do banco de dados desenvolvida;
- d) Capítulo 5, resultados experimentais de um protótipo instalado numa torneira residencial são apresentados;
- e) Apêndice A, apresenta o código do *firmware* desenvolvido para o Arduino UNO;
- f) Apêndice B, contém o código utilizado no servidor do sistema desenvolvido.

## 2 REVISÃO BIBLIOGRÁFICA

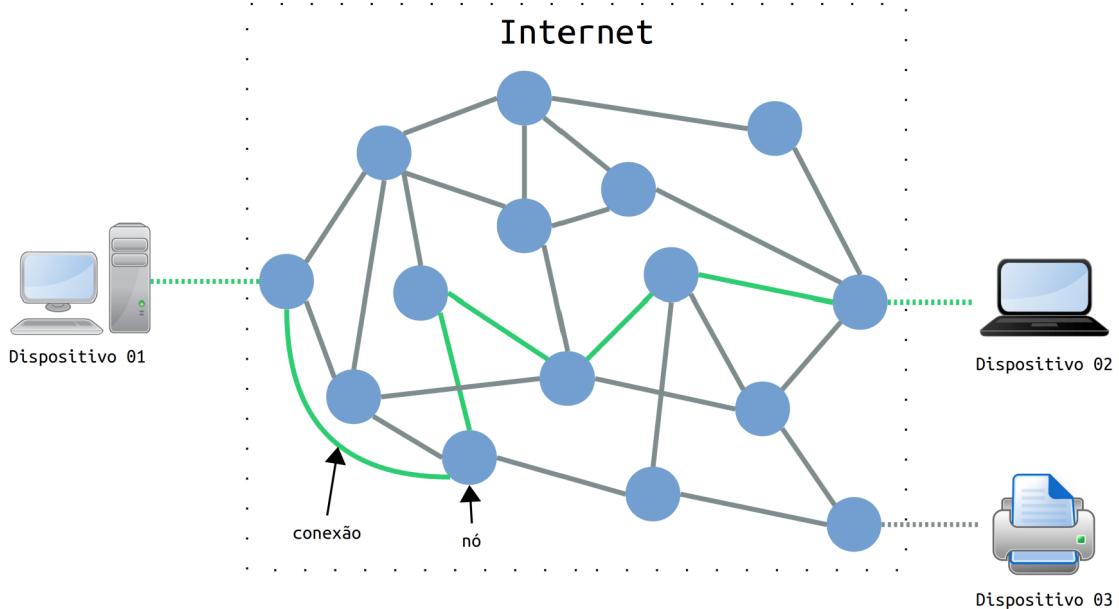
Antes de dissertar sobre a solução proposta por este trabalho se faz necessário rever alguns temas para situar o leitor e facilitar a compreensão do restante do documento.

### 2.1 A Internet e a Web

Embora a internet atualmente seja algo tão presente no nosso dia a dia que raramente se pensa como ela funciona, é necessário uma breve explicação para poder tornar trivial a compreensão da conectividade entre dispositivos.

A internet é uma rede distribuída que conecta, de ponta a ponta, todos os dispositivos presentes nela, ou seja, a rede deve ser capaz de sempre achar um caminho para que a informação flua entre dois aparelhos quaisquer. Na Figura 2 pode-se ver uma representação.

Figura 2 - Representação gráfica da internet, com uma possível conexão entre dois dispositivos destacada em verde



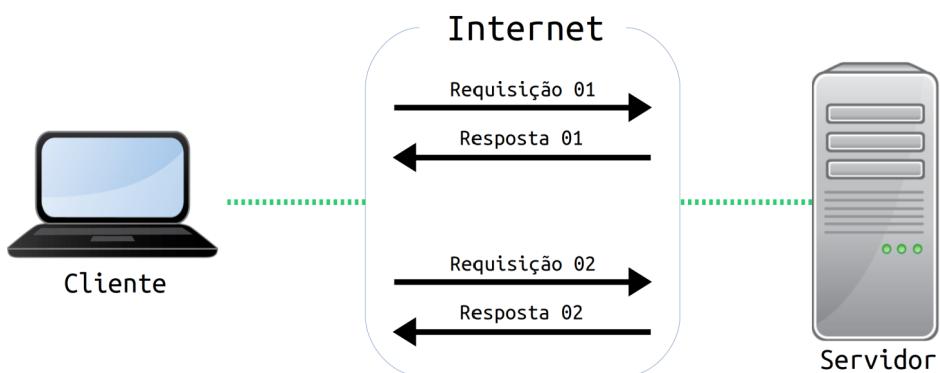
Fonte: Elaborada pelo autor.

Então, para um dispositivo se comunicar com outro na internet existe um caminho físico que a informação deve percorrer, que geralmente é uma mistura de sinais viajando num fio e pelo espaço. Essa interoperabilidade é possível graças aos protocolos desenvolvidos no início da internet, sendo o mais conhecido até hoje o *Hypertext Transfer Protocol* (HTTP),

em português Protocolo de Transferência de Hipertexto. Como o nome já diz é um protocolo baseado em texto, principal informação que circulava no início da internet.

Segundo Pfister (2011, p. 30), o HTTP funciona através de pares de requisição e resposta: o cliente, normalmente o navegador *web* de um dispositivo, faz uma **requisição** ao servidor, normalmente um computador que hospeda um *site*, e esse envia uma **resposta**. Uma representação do protocolo pode ser vista na Figura 3.

Figura 3 - Protocolo HTTP representado graficamente com seus pares de requisição e resposta



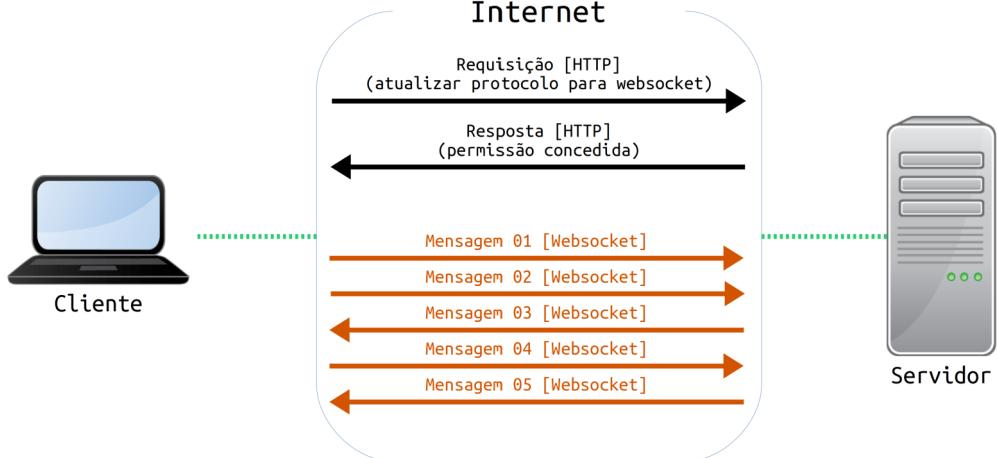
Fonte: Elaborada pelo autor.

O exemplo mais comum do uso desse protocolo é um navegador *web* de um computador requerendo uma página *web* a um servidor e esse respondendo com o envio dos arquivos que compõe a página em questão. Embora o HTTP seja um bom protocolo pela sua simplicidade e confiabilidade ele não é indicado para sistemas em tempo real pela *web*. Para esses casos o protocolo que se encaixa mais é o *WebSocket* pelos seguintes motivos:

- a) após a conexão estabelecida entre cliente e servidor não é necessário o par requisição e resposta, ficando livre ambos os lados para mandar mensagens quando necessário;
- b) diferente do HTTP que possui um cabeçalho de centenas de *bytes* o do *WebSocket* não contém mais que 3 *bytes*;
- c) não havendo necessidade de abrir uma nova conexão a cada par requisição e resposta a latência da transmissão diminui.

Uma representação gráfica do protocolo pode ser vista na Figura 4.

Figura 4 - Representação gráfica do *WebSocket*  
Internet



Fonte: Elaborada pelo autor.

A *Web*, ou *World Wide Web*, é o pequeno pedaço da internet que usa *Uniform Resource Locator* (URL), também conhecidos como endereços dos *websites* (Ex: [www.exemplo.com](http://www.exemplo.com)), para localizar servidores e através de HTTP enviar e receber informações em *HyperText Markup Language* (HTML). Navegadores *web* então renderizam arquivos HTML colocando o resultado na janela do navegador. Portanto ao acessar o *site* de uma rede social pelo navegador *web* de um computador se está utilizando da *Web* e quando a mesma rede social é acessada por um aplicativo de celular se está utilizando da internet.

## 2.2 Aplicações *Web*

Embora a linha que diferencie um *website* de um *webapp*, ou aplicação *web*, seja tênue pode-se dizer que um *website* é um *site* com páginas estáticas e um *webapp* um *site* com páginas dinâmica, assemelhando-se a um *software* comum.

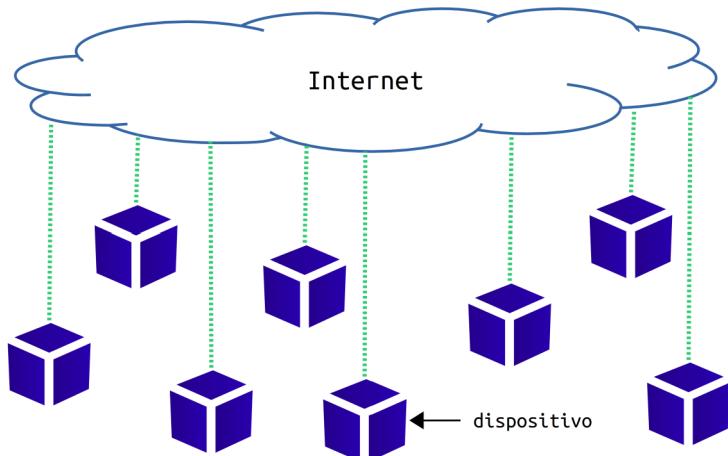
*Webapps* são mais conhecidos por permitirem que usuário tenha agência através do navegador, movendo coisas na tela e inserindo informações novas enquanto a página se atualiza em tempo real. Atualmente muitas aplicações tem surgido para substituir o trabalho realizado por *softwares* em computadores e aplicativos em celulares, como: processadores de texto, criação de apresentações, criação de planilhas, ferramentas para desenho e jogos.

## 2.3 Conectividade no panorama da Internet das Coisas

Dado que um ou mais dispositivos, com seus sensores e atuadores, podem estar monitorando o mesmo local as duas principais formas de conectá-los à internet são:

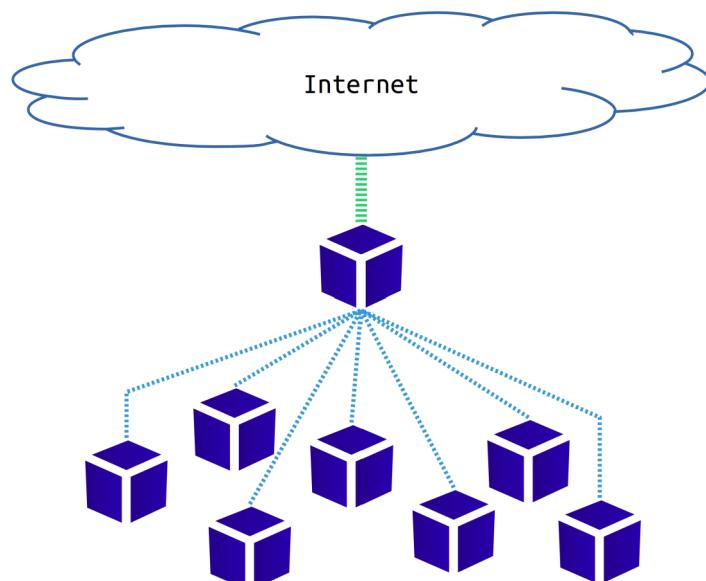
- a) cada nó, ou dispositivo, possui capacidade de se conectar diretamente a internet através do protocolo da internet, ou *Internet Protocol* (IP), possuindo cada nó um endereço IP (Figura 5);
- b) vários nós se comunicam com um nó principal que possui acesso a internet, existindo apenas um canal para a informação sair da rede local (Figura 6).

Figura 5 - Conectando os dispositivos à internet quando todos possuem capacidade IP



Fonte: Elaborada pelo autor.

Figura 6 - Conectando os dispositivos à internet quando apenas um possui capacidade IP



Fonte: Elaborado pelo autor.

Para dispositivos com capacidade IP os meios mais utilizadas para conectá-los à internet são:

- a) *Ethernet*: conexão a cabo muito conhecida por servir de estrutura para redes locais, em inglês *Local Area Networks* (LAN), é também mais segura do que as conexões sem fio. O sensor é conectado a um modem; esse conectado a um Provedor de Serviço de Internet, em inglês *Internet Service Provider* (ISP); através do cabo *ethernet*;
- b) Wi-Fi: conexão sem fio baseada na norma IEEE 802.11. Normalmente um modem/roteador Wi-Fi serve de ponto de acesso ao estabelecer uma rede sem fio local na qual o sensor se conecta;
- c) GSM/3G/4G: gerações diferentes das redes de conexão sem fio de longo alcance normalmente utilizadas pelos celulares;
- d) Satélite: um link direto do dispositivo com um ou mais satélites. Devido ao alto custo envolvido esse tipo de conexão é normalmente utilizada em regiões remotas aonde outras opções não estão disponíveis, como monitoramento de oleodutos.

Quando numa rede de sensores sem fio, em inglês *Wireless Sensor Network* (WSN), apenas um nó possui capacidade IP várias tecnologias são utilizadas para estabelecer a comunicação entre os dispositivos, entre elas podemos citar:

- a) *Bluetooth*: conexão sem fio de curto alcance bastante conhecida devido a sua presença em celulares, *tablets* e *notebooks*. Existe também o *Bluetooth Low Energy* (BLE), que busca reduzir o consumo de energia mantendo o mesmo alcance;
- b) 6LoWPAN: acrônimo para *IPv6 over Low power Personal Area Networks*. Segundo Hersent, Boswarthick e Elloumi (2012, p. 195); bastante conhecida e adotada no setor de IoT pois utiliza a nova versão do protocolo da internet (IPv6), estabelecendo redes capazes de usar o IP com baixo consumo de energia;
- c) ANT: desenvolvido pela empresa ANT Wireless, se assemelha com BLE mas com um foco maior em soluções que necessitam de baixa taxa de transferência;
- d) outras: Weightless, NB-IoT, RPMA, DASH7, Z-WAVE, Zigbee etc.

Uma lista completa de tecnologias para redes de sensores sem fio seria um trabalho árduo, se não impossível, devido a quantidade de soluções patenteadas e o surgimento de novas a cada dia.

### 3 PROJETO DO MEDIDOR INTELIGENTE E *FIRMWARE*

O dispositivo projetado para medir o consumo de água e enviar dados ao servidor, aqui chamado de medidor inteligente, é constituído de 3 componentes principais, cada um estudado em detalhes nos próximos subtópicos.

#### 3.1 Sensor de vazão de água

O sensor de vazão escolhido foi o modelo SEN-HZ21WA da SAIER por seu baixo custo e disponibilidade no mercado local, esse pode ser visto na Figura 7.

Figura 7 - Sensor de fluxo utilizado



Fonte: Elaborada pelo autor.

Seu funcionamento se baseia no efeito Hall, ou seja, o fluxo de água que atravessa o dispositivo será medido através de pulsos de tensão elétrica. O fenômeno é possível graças a presença de um magnético e uma hélice na estrutura interna do dispositivo, uma visão interna de um modelo similar pode ser vista na Figura 8.

Figura 8 - Visão interna de um sensor de vazão similar



Fonte: Hareendran [2015?].

Algumas especificações técnicas do sensor podem ser vistas na Tabela 1.

Tabela 1 - Algumas especificações técnicas do sensor de vazão

Nome do parâmetro	Valor
Pressão hidrostática máxima	1,75 MPa
Variação de fluxo	1 – 30 L/min
Tensão de operação	3 – 18 VDC
Corrente de operação máxima	15 mA
Resistência dielétrica	> 100 MΩ
Características dos pulsos por fluxo	[4,1Q] ± 10%
Dimensões	6,2 cm x 3,6 cm x 3,5 cm

Fonte: SAIER [201-?].

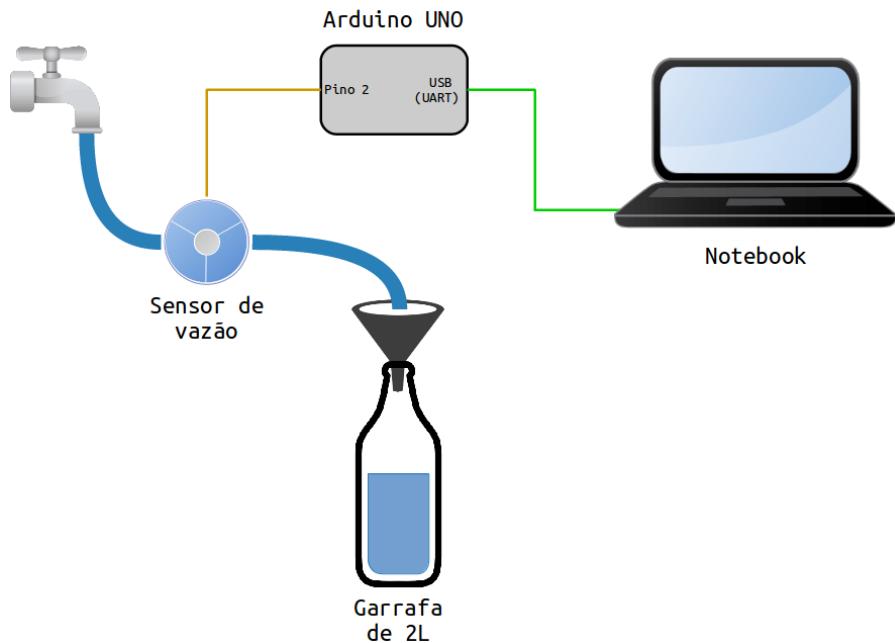
Nota-se na Tabela 1 o campo “Características dos pulsos por fluxo”, que quer dizer que a frequência dos pulsos emitidos é igual a 4,1 vezes a vazão volumétrica Q com erro de até 10%, ou seja:

$$f = 4,1Q \quad (1)$$

Supondo que as unidades da equação 1 sejam as do Sistema Internacional de Unidades, ou *Système International d'unités* (SI), a frequência é descrita em hertz (Hz) e a vazão volumétrica em m<sup>3</sup>/s, mas isso acaba gerando uma inconsistência. Através de uma regra de três calcula-se que o sensor emitiria um pulso de tensão a cada 243,9 L de água que passassem por ele, sendo incoerente com o tamanho e capacidade de vazão do componente. A possível causa desse contrassenso é que a unidade da vazão volumétrica não é m<sup>3</sup>/s, então decidiu-se realizar testes em vez de buscar a unidade correta porque essa seria irrelevante sem a confirmação através de dados experimentais.

O experimento constituiu de uma garrafa de 2 L preenchida com água que atravessava o sensor, esse conectado a um Arduino UNO que detectou através de um pino com interrupção externa as bordas de descida dos pulsos de tensão emitidos pelo sensor. O Arduino foi também conectando, através de uma porta USB, a um notebook para poder comunicar em tempo real a quantidade acumulada de pulsos. Uma representação gráfica do experimento proposto pode ser vista na Figura 9.

Figura 9 - Teste proposto para calibração do sensor de vazão



Fonte: Elaborada pelo autor.

A garrafa de 2 L foi preenchida 5 vezes e os resultados podem ser vistos na Tabela 2.

Tabela 2 - Resultados dos testes para a calibração do sensor de vazão

Número do teste	Quantidade de pulsos medidos
1º	1202
2º	1365
3º	1338
4º	1328
5º	1350

Fonte: Elaborada pelo autor.

Fazendo uma média aritmética dos dados obtidos tem-se que aproximadamente 1316,6 pulsos equivalem a 2 L de água, ou seja, cada pulso equivale a 1,519 mL de água. Essa constante foi então utilizada pelo servidor para converter os números de pulsos em volume de água e em posse da data e hora dos registros de consumo o servidor foi também capaz de calcular a vazão de água. Descoberta a razão entre mL e pulsos pôde-se calcular as frequências mínimas e máximas do sinal do sensor, sabendo que o componente opera com vazões de 1 a 30 L/min tem-se uma frequência mínima de 10,97 Hz e máxima de 329,16 Hz.

### 3.2 Arduino UNO

O Arduino UNO é uma plataforma de prototipagem *open-source* cujo coração é um microcontrolador ATmega328P da Atmel, cujas especificações técnicas podem ser vistas na Tabela 3.

Tabela 3 - Especificações técnicas do microcontrolador ATmega328P

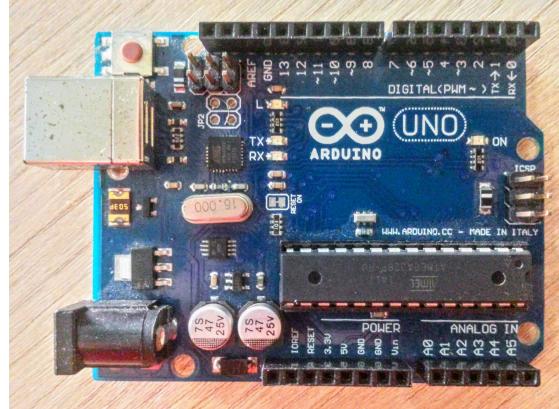
Nome do parâmetro	Valor
Tipo de processador	8-bit
Tipo de memória para programa	<i>Flash</i>
Tamanho da memória para programa	32 KB
Velocidade da CPU	20 MIPS
Tamanho da memória RAM	2 <i>bytes</i>
Tamanho da memória EEPROM	1024 <i>bytes</i>
Periféricos de comunicação digital	x1 UART, x2 SPI, x1 I2C
PWM	6
Comparadores	1
Temperatura de operação	-40 °C a +85 °C
Tensão de operação	1,8 V a 5,5 V
Número de pinos	32

Fonte: MICROCHIP (2016).

Além de oferecer um *hardware* de fácil manuseio para rápida prototipagem ainda é proporcionado uma IDE com uma linguagem de programação mais simples baseada em *Wiring*, segundo ARDUINO [201-?].

Conforme o *site* oficial do ARDUINO [201-?], a plataforma possui um grande público devido a sua natureza *open-source* e seus usuários costumam publicar seus projetos pessoais com a mesma licença, viabilizando um extenso repositório de projetos. A escolha dessa plataforma foi feita devido a grande quantidade de conhecimento disponível de graça em torno desse dispositivo, o que facilitou o desenvolvimento da solução proposta por este trabalho de conclusão de curso. Um Arduino UNO pode ser visto na Figura 10.

Figura 10 - Arduino UNO



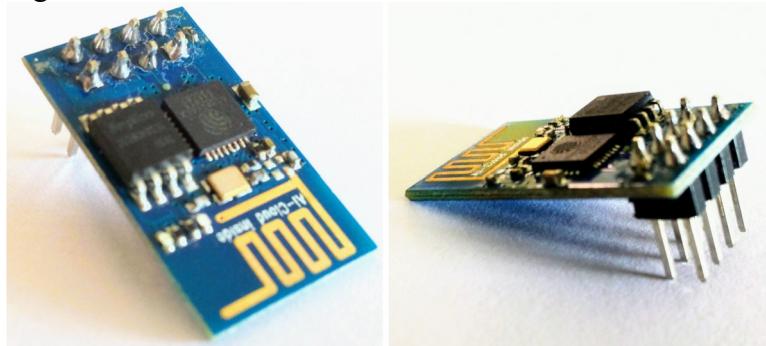
Fonte: Elaborada pelo autor.

No projeto o Arduino ficou responsável por detectar os pulsos emitidos pelo sensor de vazão de água através de um pino com interrupção externa sensível à bordas de descida, construir as requisições HTTP e enviar os comandos necessários ao módulo Wi-Fi para que esse enviasse os dados pela internet ao servidor.

### 3.3 Módulo Wi-Fi

O ESP8266 é um circuito integrado compacto com capacidade Wi-Fi produzido pela empresa chinesa ESPRESSIF. O chip é vendido em vários formatos para atender as necessidades específicas de cada projeto, neste trabalho a versão “01” fui utilizada devido a disponibilidade do mercado local. Um ESP8266-01 pode ser visto na Figura 11.

Figura 11 - ESP8266-01



Fonte: Elaborada pelo autor.

Algumas informações técnicas do módulo podem ser vistas na Tabela 4.

Tabela 4 - Especificações técnicas do ESP8266

Nome do parâmetro	Valor
Tipo de processador	32-bit
Clock da CPU	80 MHz
Memória RAM	160 KB
Periféricos de comunicação digital	x2 UART, x2 SPI, x1 I2C, x2 I2S
Conversores analógico-digital	x1 10 bits
Tensão de operação	2,5 V a 3,6 V
Corrente média de operação	80 mA
Temperatura de operação	-40 °C a +125 °C
Protocolo Wi-Fi	802.11 b/g/n/e/i
Modos do Wi-Fi	Station/SoftAP/SoftAP+Station
Segurança	WPA/WPA2
Protocolos de rede	IPv4, TCP/UDP/HTTP/FTP

Fonte: Adaptado de Espressif (2017).

Em vez de programar no dispositivo um novo *firmware* o ESP8266-01 foi utilizado no projeto com o *firmware* já instalado pela empresa chinesa, no qual pode-se controlar o módulo Wi-Fi por comandos AT transmitidos através da UART. Segundo Developers Home [200-?], comandos AT significam “Attention”, ou “Atenção” em português, que são instruções utilizadas para controlar *modens*. Dessa forma o Arduino UNO comandou o ESP8266, fazendo-o executar requisições HTTP com o servidor. Uma lista com alguns comandos AT pode ser vista na Tabela 5.

Tabela 5 - Exemplo de comandos AT disponíveis para o ESP8266

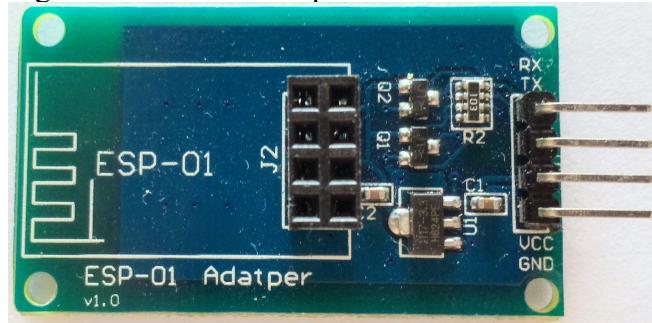
Comando	Descrição
AT	Testa comunicação por comandos AT
AT+RST	Reinicia o módulo
AT+GSLP	Entra em modo de sono profundo
AT+RESTORE	Restaura configurações de fábrica
AT+SYSADC	Checa o valor do conversor analógico-digital
AT+CWMODE	Define modo de operação do Wi-Fi ( <i>Station</i> , <i>SoftAP</i> , <i>Station + SoftAP</i> )
AT+CIPSTART	Estabelece uma conexão TCP, transmissão UDP ou conexão SSL
AT+CIPSEND	Envia dados pela internet
AT+CIPCLOSE	Encerra conexão TCP/UDP/SSL

Fonte: Adaptado de Espressif (2017).

### 3.4 Hardware do Medidor Inteligente

Ao juntar todos os componentes num circuito final surgiu um problema: enquanto o Arduino UNO trabalha com 5 V na sua alimentação e lógica o ESP8266-01 trabalha com 3,3 V, impedindo uma conexão direta entre os dois através da UART. Para resolver esse problema foi adquirido um módulo adaptador para o ESP8266, possibilitando-o ser alimentado com 5 V e realizar uma interface digital com dispositivos que trabalhem com 5 V. O módulo é composto por um regulador de tensão e conversores de nível de lógica, esse pode ser visto na Figura 12.

Figura 12 - Módulo adaptador do ESP8266-01



Fonte:Elaborada pelo autor

Alguns dados do módulo podem ser vistos na Tabela 6.

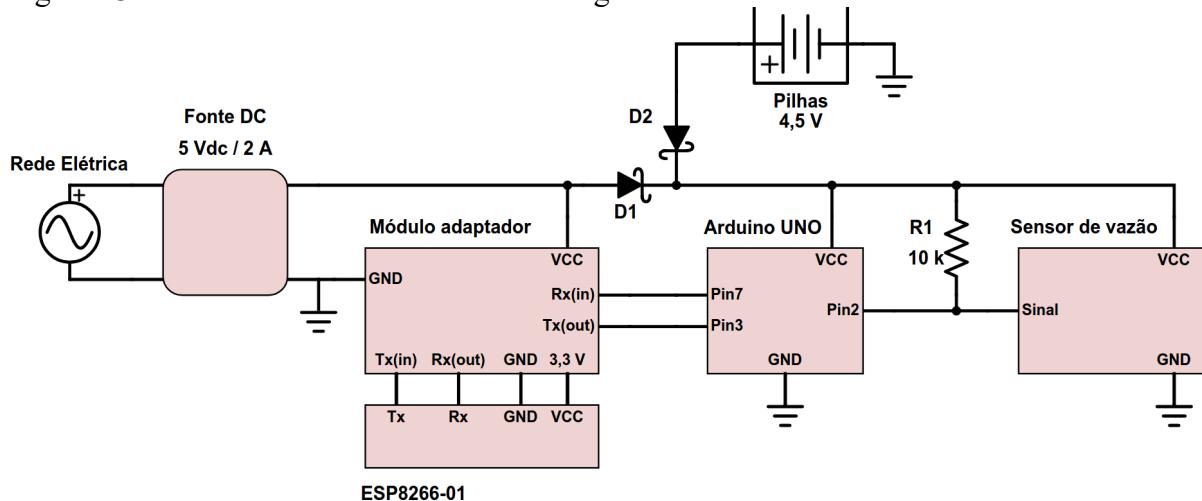
Tabela 6 - Especificações técnicas do módulo adaptador

Nome do parâmetro	Valor
Tensão de entrada	4,5 V a 5,5 V
Tensão de saída	3,3 V
Corrente de operação	0 mA a 240 mA
Tensão de interface lógica	3,3 V / 5 V

Fonte: Adaptado de Banggood [2016?].

O esquemático da Figura 13 mostra o circuito completo.

Figura 13 - Circuito final do hidrômetro inteligente



Fonte: Elaborada pelo autor.

De acordo com a Figura 13, quando a rede elétrica está presente o diodo D1 está polarizado diretamente enquanto D2 reversamente; dessa forma o ESP8266, o Arduino e o sensor de vazão estarão energizados. Quando há uma falta na rede elétrica o diodo D1 é polarizado reversamente enquanto D2 diretamente; dessa forma as pilhas alimentam apenas o Arduino UNO e o sensor de vazão. Essa fonte de emergência não fornece energia ao módulo Wi-Fi na ausência da rede elétrica porque os roteadores Wi-Fi da instalação e consequentemente o acesso a internet provavelmente não estarão funcionando, o que tornaria inútil o trabalho do ESP8266. Nessa topologia as pilhas servem para aumentar a confiabilidade do sistema de monitoramento, visto que o mesmo estará medindo o consumo mesmo na ausência da rede da concessionária de energia elétrica.

Ambos diodos do circuito, D1 e D2, são do tipo *schottky* 1N5819, cujas especificações podem ser vistas na Tabela 7. A escolha desse tipo de diodo se deu pelo fato de apresentarem uma queda de tensão menor do que a dos diodos tradicionais, garantindo o nível de tensão adequado para os componentes operarem de acordo com suas especificações.

Tabela 7 - Especificações técnicas do diodo 1N5819

Nome do parâmetro	Valor
Tensão reversa máxima	40 V
Temperatura de operação	-65 °C a 125 °C
Tensão de polarização direta ( $I_F = 1,0 \text{ A}$ )	0,4 V

Fonte: Adaptado de Alldatasheet [200-?].

Por fim um resistor de *pull-up* de  $10 \text{ k}\Omega$  foi utilizado para garantir que o sinal de

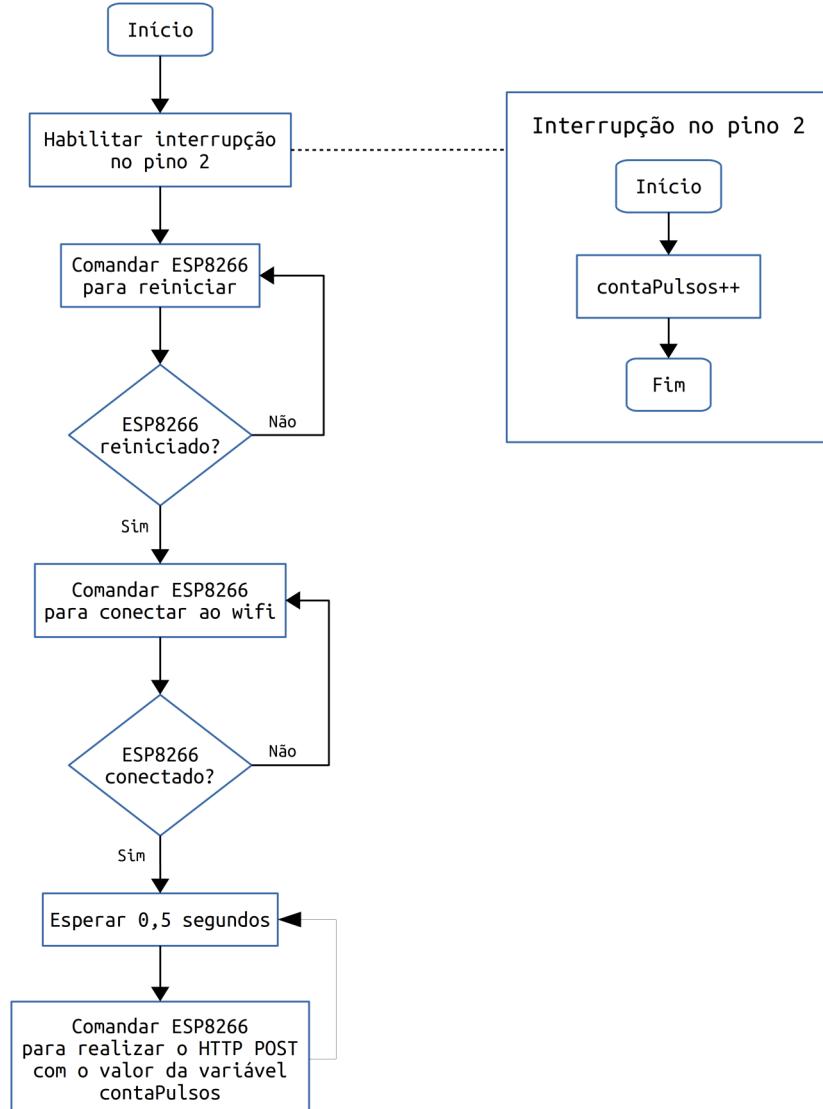
saída do sensor de vasão se mantivesse no nível lógico alto após o envio de pulsos ao Arduino.

### 3.5 Algoritmo do *firmware*

O *firmware* desenvolvido para o Arduino UNO seguiu o algoritmo descrito na Figura 14. O código completo pode ser encontrado no APÊNDICE A – FIRMWARE DO MEDIDOR INTELIGENTE.

Figura 14 - Algoritmo do *firmware*

Programa principal



Fonte: Elaborada pelo autor.

Nota-se pelo algoritmo que o medidor inteligente sempre inicia o contato com o servidor e o inverso nunca acontece. O servidor ainda é capaz de mandar informações para o dispositivo através da resposta do par requisição-resposta do HTTP, mas nunca iniciar um contato. O ESP8266 foi configurado dessa forma por medida de segurança, prevenindo que terceiros accessem o medidor inteligente.

## 4 APLICAÇÃO WEB

Para uma aplicação *web* existir há um programa, ou um conjunto deles, trabalhando num servidor. O servidor então fica responsável por se comunicar com os clientes, gerenciar o banco de dados, processar e gerar *scripts* em HTML, enviar e-mails entre outras atividades.

Na solução apresentada neste trabalho a linguagem de programação no lado do servidor foi *Python*, utilizando a *framework Flask*. O *Flask* foi escolhido por ser bastante leve e modular, permitindo ao desenvolvedor a criação de uma aplicação enxuta e veloz.

Conforme Grinberg (2014), para acelerar o desenvolvimento essa *framework* possui bibliotecas chamadas de “extensões” que auxiliam na adição de funcionalidades ao sistema. Para deixar mais claro segue abaixo a lista de extensões utilizadas no projeto:

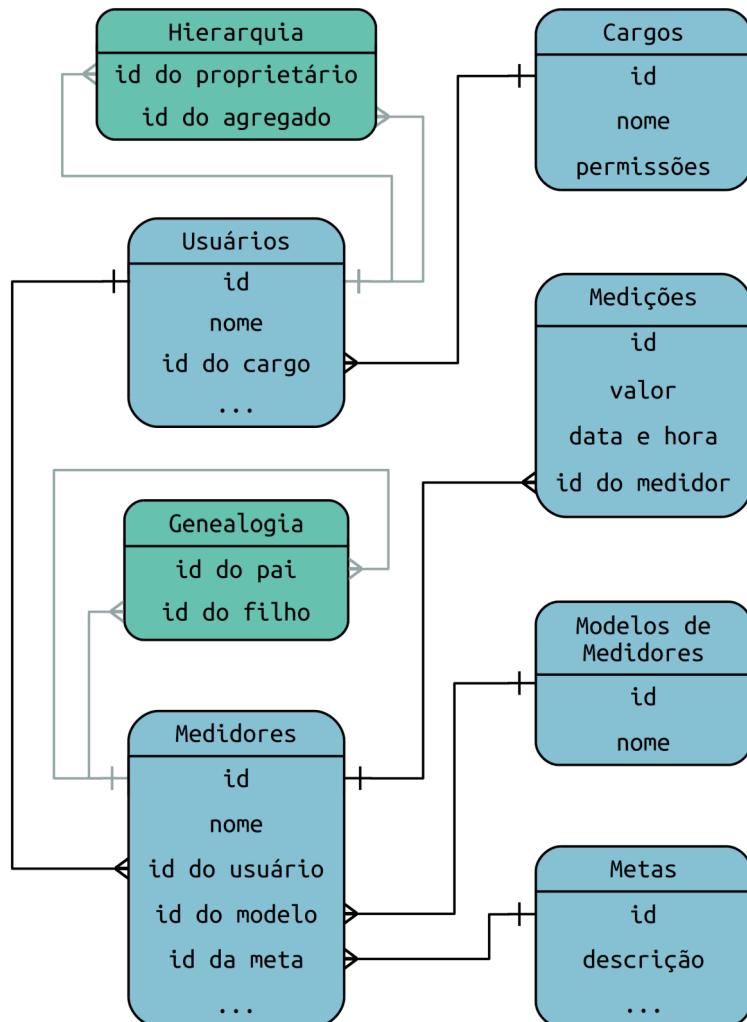
- a) *Flask-Script*: permite escrever e rodar *scripts* fora do *webapp* como testes, configuração e inicialização do banco de dados e de um *shell*;
- b) *Flask-WTF*: integra *Flask* com *WTForms* para a criação de formulários;
- c) *Flask-SQLAlchemy*: inclui suporte para um conjunto de ferramentas em *Python* para SQL, linguagem utilizada para bancos de dados relacionais, conhecida como *SQLAlchemy*;
- d) *Flask-Migrate*: adiciona o recurso de migrações do banco de dados utilizando *SQLAlchemy*, *Alembic* e *Flask-Script*;
- e) *Flask-SocketIO*: integra ao *Flask* a biblioteca para clientes *SocketIO*, permitindo o uso de *WebSockets* na aplicação;
- f) *Flask-Mail*: torna bem mais simples a tarefa de desenvolver uma aplicação que envie e-mails.

### 4.1 Design do banco de dados

Numa solução que se enquadra na Internet das Coisas uma parte crucial do projeto é o banco de dados; nele será guardado todas as informações que as “coisas” achem relevante enviar ao servidor e outras informações necessárias para o funcionamento do sistema. O design do banco de dados apresentado neste tópico contém espaço para mais recursos que os oferecidos pela solução deste trabalho, devido à falta de tempo a implementação completa fica a cargo de trabalhos futuros.

Um diagrama do esquema do banco de dados pode ser visto na Figura 15, nele é possível ver que todas as relações entre tabelas são do tipo “um para muitos”, ou seja, um usuário pode ter vários medidores mas um medidor pode ter apenas um usuário. O lado “um” da relação é representado por um traço vertical e o “muitos” por um “pé de galinha”.

Figura 15 - Esquema do banco de dados



Fonte: Elaborada pelo autor.

As tabelas e as conexões de “Hierarquia” e “Genealogia” estão destacadas por serem tabelas auxiliares. Nos próximos subtópicos cada tabela será descrita em detalhes.

#### 4.1.1 Tabela “Usuários”

A tabela “Usuários” guarda todas as informações relevantes de cada usuário da plataforma. Possui as colunas:

- a) ID: número de identificação do usuário;
- b) Nome: nome completo do usuário da aplicação *web*;
- c) Endereço: endereço completo do usuário;
- d) E-mail: e-mail válido do usuário em questão;
- e) Senha: *hash* da senha criada pelo usuário;
- f) ID do Cargo: número de identificação do cargo do usuário;
- g) Confirmado: booliano que indica se o usuário clicou no link de confirmação enviado pelo e-mail ou não.

#### **4.1.2 Tabela “Cargos”**

A tabela “Cargos” contém os cargos com suas respectivas permissões no *webapp*.

Possui as colunas:

- a) ID: número de identificação do cargo;
- b) Nome: com os possíveis valores:
  - Proprietário: usuário que possui um ou mais medidores;
  - Agregado: usuário que não possui nenhum medidor mas tem acesso a medidores de outro usuário. Ex: pais cedendo permissão aos filhos e empresários cedendo permissão aos funcionários;
  - Administrador: usuário com todas as permissões do *webapp*;
- c) Permissões: visto que os cargos não foram utilizados na versão final do projeto esse espaço ficou em branco.

#### **4.1.3 Tabela “Medidores”**

A tabela “Medidores” engloba as informações relevantes sobre os medidores inteligentes do sistema. Possui as colunas:

- a) ID: número de identificação do medidor;
- b) Nome: nome dado ao medidor para facilitar sua identificação. Ex: “Casa”, “Chuveiro da empregada”, “Banheiro dos professores” etc;
- c) Endereço: endereço da instalação do medidor, já que este pode ser diferente do local do proprietário;
- d) Preço Médio: preço médio em R\$/m<sup>3</sup> da água que passa pelo medidor;
- e) Cte: constante de conversão do sensor em m<sup>3</sup>/pulso;

- f) ID do Usuário: número de identificação do usuário proprietário do medidor;
- g) ID do Modelo do Medidor: número de identificação do modelo do medidor;
- h) ID da Meta: número de identificação da meta do medidor.

#### **4.1.4 Tabela “Modelos de Medidores”**

Essa tabela foi criada para que no futuro, quando houver várias categorias e versões de medidores, novos modelos sejam adicionados nela assim como novas colunas contendo os recursos e capacidades de cada um. Atualmente só possui duas colunas:

- a) ID: número de identificação do modelo de medidor;
- b) Nome: identificação do modelo como. Ex: “1.0”, “Industrial 2.3” etc.

#### **4.1.5 Tabela “Metas”**

A tabela “Metas” foi criada porque um dos principais objetivos do sistema é ajudar o proprietário a reduzir a conta de água, para isso uma atitude comum é estabelecer uma meta de consumo para cada medidor inteligente. O usuário pode então avaliar no *webapp* se está alcançando a redução desejada assim como ser notificado por e-mail sobre o consumo. A tabela “Metas” possui as colunas:

- a) ID: número de identificação da meta;
- b) Descrição: uma explicação breve do porquê da meta. Ex: “Evitar pagar taxa de contingência”;
- c) Valor: o número que representa a meta. Ex: “22” de uma meta de “22 m<sup>3</sup>”;
- d) Unidade do Valor: pode assumir os valores “m<sup>3</sup>”, “L” ou “R\$”;
- e) Início: data e hora do início da meta;
- f) Intervalo: duração de tempo da meta. Normalmente assume intervalos de um dia, uma semana ou um mês;
- g) Notificações: intervalo de tempo entre envios de relatórios sobre o andamento do consumo por e-mail;
- h) Notificado: número binário que indica se o usuário já foi notificado, ou seja, já recebeu um e-mail informando sobre o estado da meta. Uma notificação ocorre quando se atinge 80% ou 100% da meta diária ou mensal.

#### **4.1.6 Tabela “Medições”**

A tabela “Medições” guarda todas as medições enviadas de todos os medidores. Possui as colunas:

- a) ID: número de identificação da medição;
- b) Valor: número acumulado de pulsos do sensor;
- c) Data e Hora: data e hora que o servidor recebeu o dado do dispositivo;
- d) ID do Medidor: número de identificação do medidor que enviou a medição.

#### **4.1.7 Tabela “Hierarquia”**

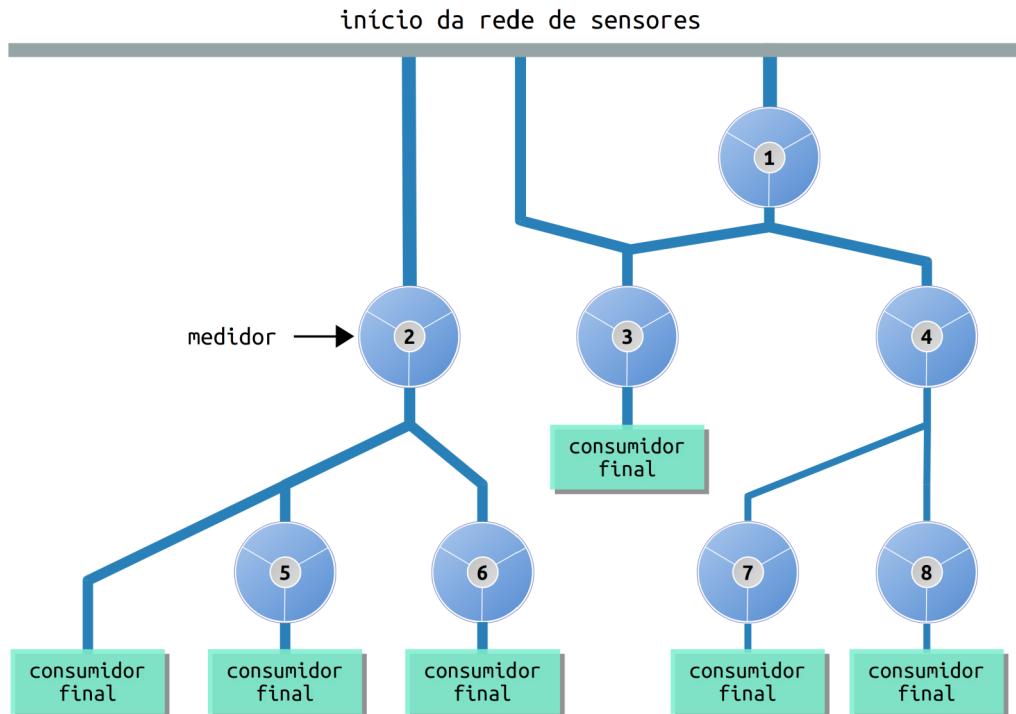
Visto que pode ser concedida a permissão a um ou mais usuários (agregados) de verem as informações de medidores de outro usuário (proprietário) há a necessidade de uma nova tabela que descreva essa relação: a tabela “Hierarquia”. Essa tabela auxiliar possui as colunas:

- a) ID do proprietário: número de identificação do usuário que concedeu a permissão;
- b) ID do agregado: número de identificação do usuário que recebeu a permissão.

#### **4.1.8 Tabela “Genealogia”**

Alguns dos objetivos do sistema são ter uma visão seccionada da instalação e a identificação de vazamentos, para isso ocorrer uma rede de sensores deve ser implementada. A Figura 16 contém um exemplo de uma rede de hidrômetros inteligentes.

Figura 16 - Rede de hidrômetros inteligentes



Fonte: Elaborada pelo autor.

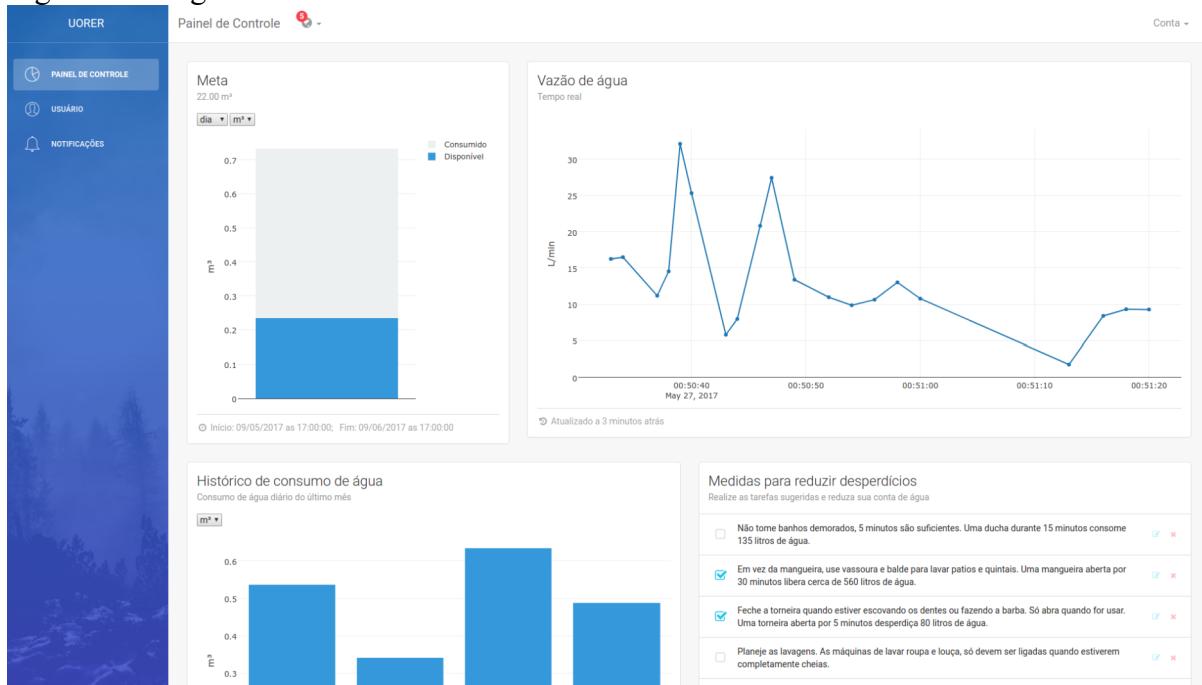
Assemelhando-se à uma árvore genealógica o esquema da Figura 16 também representa uma Infraestrutura Avançada de Medição, em inglês *Advanced Metering Infrastructure* (AMI), permitindo identificar vazamentos entre os medidores 4, 7 e 8 porque a conexão entre eles é completa, ou seja, todos os “pais” e “filhos” são conhecidos. Nas tubulações entre os medidores 1, 3 e 4 e entre 2, 5 e 6 não podem ser detectados vazamentos pois, respectivamente, há ausência de um medidor pai e de um medidor filho, o máximo que pode ser feito nessa situação é inferir o consumo do elemento restante. A tabela então possui as colunas:

- a) ID do pai: número de identificação do medidor que está acima, na relação;
- b) ID do filho: número de identificação do medidor que está abaixo, na relação.

#### 4.2 O site

A interface do usuário com o sistema é o *site* da *Web*. Esse foi desenvolvido para ser responsivo, ou seja, ele se adapta ao tamanho da tela permitindo ao usuário interagir com uma interface gráfica agradável mesmo usando um *tablet* ou *smartphone*. A página inicial, e única, do *site* pode ser vista na Figura 17. Apesar da página conter uma barra superior e lateral esquerda com botões, eles não foram implementados.

Figura 17 - Página inicial do site

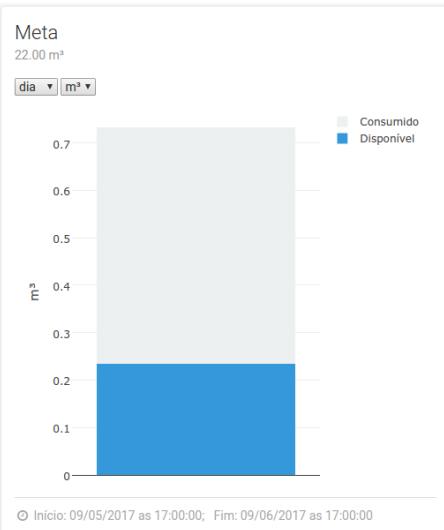


Fonte: Elaborada pelo autor.

#### 4.2.1 Meta

Essa seção apresenta o consumo em relação a meta, conforme Figura 18.

Figura 18 - Seção da página web intitulada "Meta"



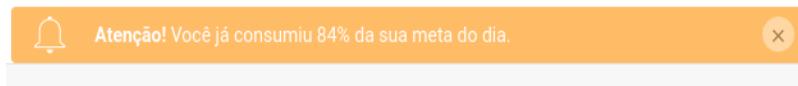
Fonte: Elaborada pelo autor.

Logo abaixo do título consta a meta estabelecida pelo usuário numa cor acinzentada, nesse caso 22,00 m<sup>3</sup>, seguida de dois menus suspensos, na figura selecionadas as

opções “dia” e “m<sup>3</sup>”. O primeiro menu suspenso possui as opções “dia” e “mês” para o usuário escolher em que fração de tempo quer ver informações e o segundo contém “m<sup>3</sup>”, “L” e “R\$” como opções para escolher a unidade em que as informações são exibidas. Posterior a esses elementos existe a representação de um tanque de água cuja capacidade máxima é a meta em questão, em cinza o volume de água consumido e em azul o que ainda está disponível. Ao passar o *mouse* sobre o tanque, ou clicá-lo no caso de um aparelho *touchscreen*, os valores “Consumido” e “Disponível” aparecem flutuando assim como suas porcentagens em relação à meta.

Como se trata de um elemento que se atualiza em tempo real, ou seja, toda vez que um dado chega ao servidor ele é processado e os novos valores de “Consumido” e “Disponível” são enviados ao navegador *web* do cliente, caso o consumo ultrapasse 80% da meta diária ou mensal uma notificação aparecerá no canto superior direito da página *web*. A Figura 19 inclui um exemplo da notificação.

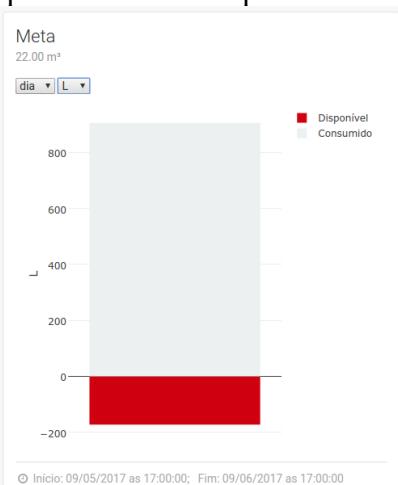
Figura 19 - Notificação na página *web* da ultrapassagem de 80% da meta diária



Fonte: Elaborada pelo autor.

Uma notificação de ultrapassagem de 100% assume o mesmo formato mas com a cor vermelha. Além da notificação ao consumir mais de 100% o tanque de água sofre modificações conforme a Figura 20.

Figura 20 - Gráfico da meta quando essa é ultrapassada



Fonte: Elaborada pelo autor.

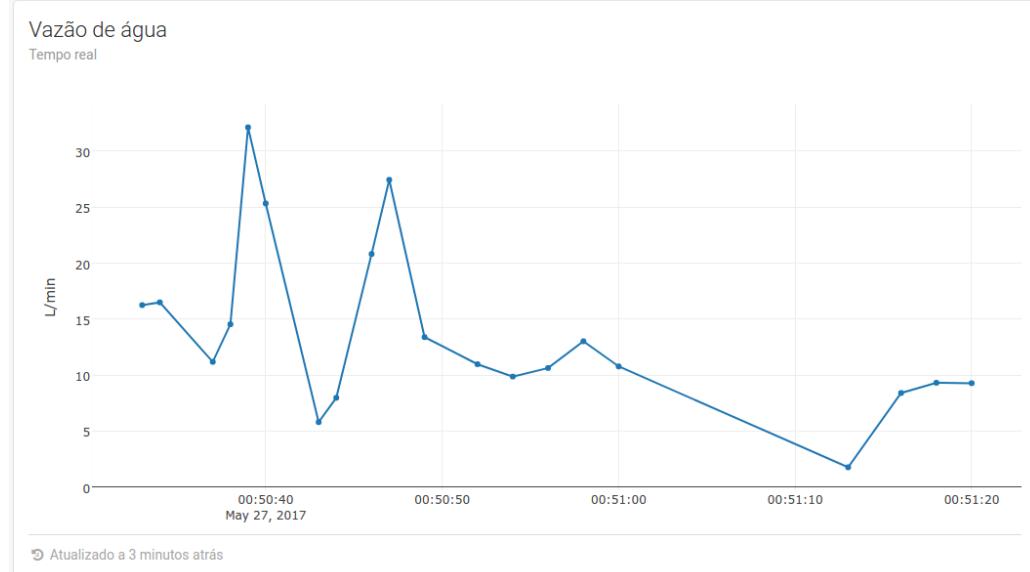
Pode-se notar que “Disponível” assume a coloração vermelha e valores negativos para destacar a situação atual de consumo.

O último elemento dessa seção, a nota de rodapé, contém a data e hora do início e do fim do período estipulado para a meta.

#### **4.2.2 Vazão de água**

Essa seção apresenta em tempo real a vazão de água que passa pelo medidor, conforme a Figura 21.

Figura 21 - Seção que apresenta a vazão de água em tempo real



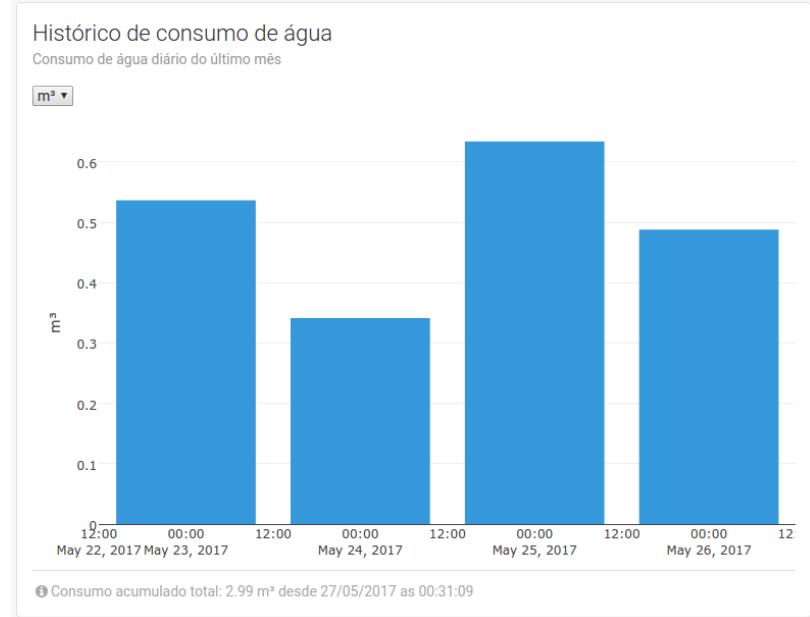
Fonte: Elaborada pelo autor.

Entre suas utilidades vale destacar a identificação de vazamentos, quando é registrada alguma vazão diferente de zero mesmo quando o usuário verificou que todos os pontos de acesso a água estão fechados.

#### **4.2.3 Histórico de consumo de água**

A seção “Histórico de consumo de água” fornece um gráfico que permite acompanhar o consumo diário de água do último mês, visto em detalhes na Figura 22, possibilitando a identificação de eventos que tenham impactado no consumo de água da instalação.

Figura 22 - Seção que mostra o histórico de consumo de água



Fonte: Elaborada pelo autor.

#### 4.2.4 Medidas para reduzir desperdícios

A última seção da página contém dicas para auxiliar um cliente residencial à reduzir o consumo de água através da redução de desperdícios. É possível marcar num checkbox a esquerda as medidas que já foram tomadas. Uma vista mais detalhada dessa porção do site está presente na Figura 23.

Figura 23 - Seção intitulada "Medidas para reduzir desperdícios"

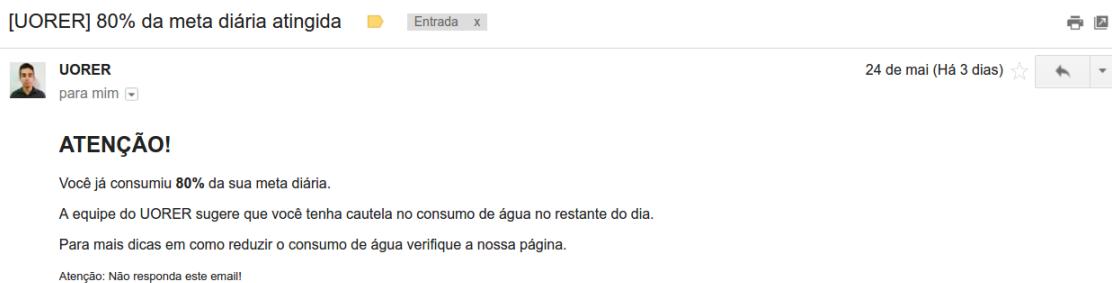
Medidas para reduzir desperdícios	
Realize as tarefas sugeridas e reduza sua conta de água	
<input type="checkbox"/>	Não tome banhos demorados, 5 minutos são suficientes. Uma ducha durante 15 minutos consome 135 litros de água. <span style="float: right;">🕒 ✎</span>
<input checked="" type="checkbox"/>	Em vez da mangueira, use vassoura e balde para lavar patios e quintais. Uma mangueira aberta por 30 minutos libera cerca de 560 litros de água. <span style="float: right;">🕒 ✎</span>
<input checked="" type="checkbox"/>	Feche a torneira quando estiver escovando os dentes ou fazendo a barba. Só abra quando for usar. Uma torneira aberta por 5 minutos desperdiça 80 litros de água. <span style="float: right;">🕒 ✎</span>
<input type="checkbox"/>	Planeje as lavagens. As máquinas de lavar roupa e louça, só devem ser ligadas quando estiverem completamente cheias. <span style="float: right;">🕒 ✎</span>
<input type="checkbox"/>	Reaproveite a água da sua máquina de lavar roupas para lavar a calçadas e dar descarga em vazos sanitários. <span style="float: right;">🕒 ✎</span>
<input type="checkbox"/>	Cheque se há gotejamento em alguma torneira e chuveiro na casa <span style="float: right;">🕒 ✎</span>
<span style="font-size: small;">🕒 Atualizado a 3 minutos atrás</span>	

Fonte: Elaborada pelo autor.

### 4.3 Outros recursos

Além da página *web* o sistema tem capacidade de enviar e-mails ao usuário quando este atingir 80% e 100% da sua meta diária ou mensal. Dessa forma não há necessidade de ficar acessando o *site* para saber se a meta está sendo respeitada. Um exemplo pode ser visto na Figura 24.

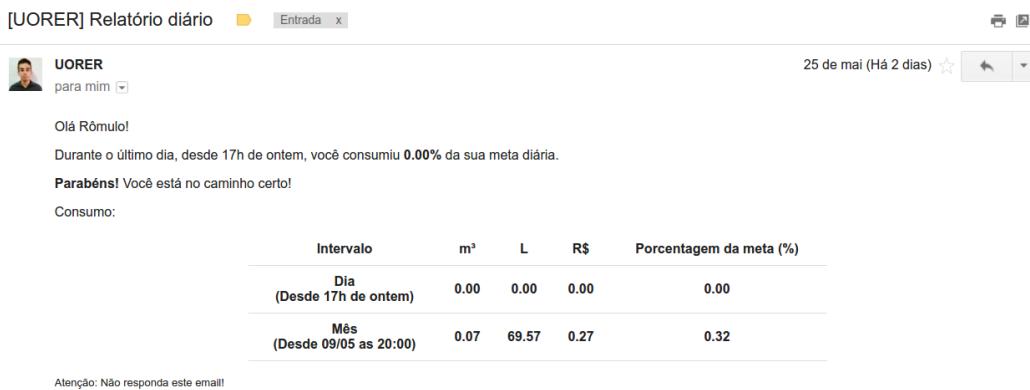
Figura 24 - E-mail notificando o consumo de 80% da meta diária



Fonte: Elaborada pelo autor.

Outro tipo de e-mail enviado é um relatório diário de consumo, permitindo uma visão constante da conta de água sem exigir esforço do ponto de vista do usuário. Um exemplo do relatório pode ser visto na Figura 25.

Figura 25 - E-mail do relatório diário



Fonte: Elaborada pelo autor.

Ambos e-mails automáticos e outras sub-rotinas foram possíveis graças a uma extensão capaz de realizar tarefas agendadas e assincronamente: *Celery*. O aprendizado de tal extensão se deu através da documentação do *site* oficial, Solem (2017), e através do blog oficial do engenheiro de *software* Miguel Grinberg, Grinberg (2015).

## 5 RESULTADOS EXPERIMENTAIS

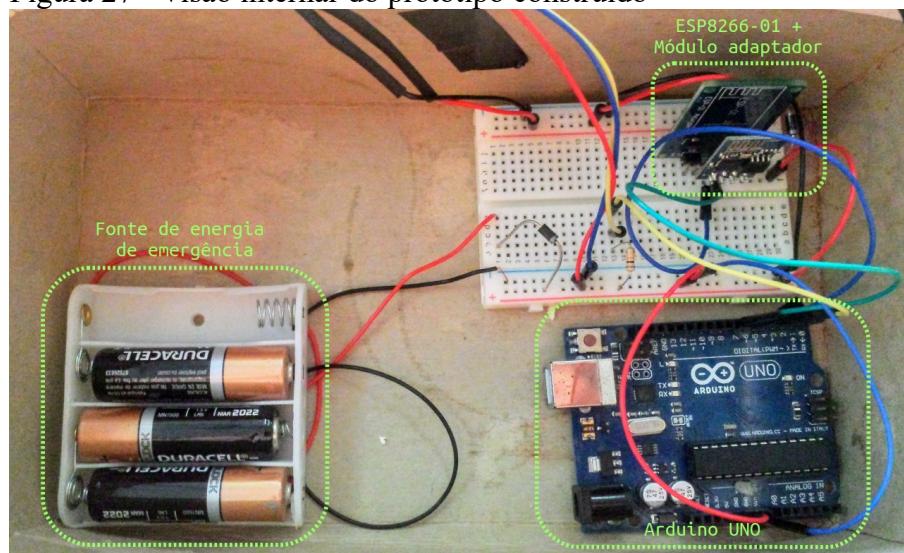
Visando testar o sistema desenvolvido o medidor inteligente foi instalado numa torneira de um banheiro residencial. Uma foto da instalação pode ser vista na Figura 26 enquanto na Figura 27 há uma visão mais detalhada do protótipo construído.

Figura 26 - Protótipo instalado



Fonte: Elaborada pelo autor.

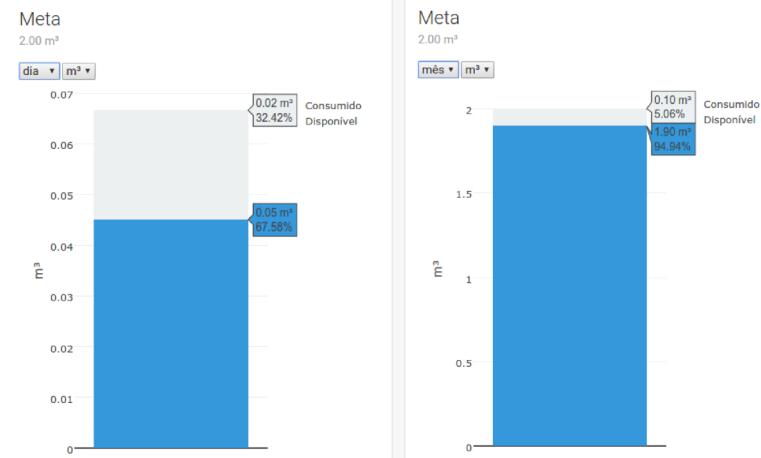
Figura 27 - Visão interna do protótipo construído



Fonte: Elaborada pelo autor.

Realizada a instalação o sistema ficou em funcionamento durante 5 dias. Uma meta de 2 m<sup>3</sup> ao mês foi estabelecida para a torneira e no 5º dia a relação entre o consumo e a meta diária e mensal pode ser vista na Figura 28.

Figura 28 - Consumo de água em relação a meta diária e mensal

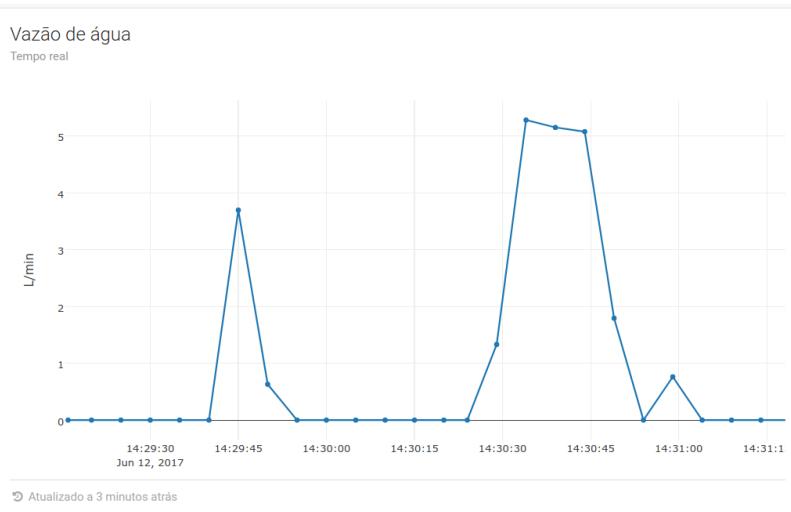


Fonte: Elaborada pelo autor.

Ao clicar na representação gráfica do tanque, na Figura 28, torna-se possível ver mais claramente os valores de água consumida e disponível, sendo a soma dos dois a própria meta. Sabendo que os gráficos foram tirados do quinto dia de teste, se a média do consumo se mantiver constante até o dia da fatura da conta água pode-se calcular, através de uma simples regra de três, que será consumido 0,62 m<sup>3</sup> de água ou 34,72% da meta.

Para testar a vazão em tempo real foi observado o comportamento do medidor na ocasião em que um usuário escovou os dentes, o resultado gráfico pode ser visto na Figura 29.

Figura 29 - Vazão em tempo real para um usuário escovando os dentes



Fonte: Elaborada pelo autor.

O primeiro pico de vazão ocorre quando o indivíduo enxágua a pasta de dente presente na escova, o segundo quando o mesmo se limpa tirando a pasta da boca e o pequeno

consumo no final se dá quando o usuário percebe que sua mão ainda está suja e a enxágua novamente.

Em nenhum dia durante o período de observação o consumo atingiu 80% da meta, não sendo portanto o usuário notificado no *site* ou por e-mail. O histórico de consumo da torneira pode ser visto graficamente na Figura 30 e com seus valores exatos na Tabela 8. O baixo consumo no dia 8 de junho ocorreu porque foi o dia em que o equipamento foi instalado, a noite.

**Figura 30 - Histórico de consumo de água pela torneira visto pelo *site***



Fonte: Elaborada pelo autor.

**Tabela 8 - Histórico de consumo da torneira em litros**

Data	Consumo
08/06/2017	3,15 L
09/06/2017	14,23 L
10/06/2017	30,18 L
11/06/2017	22,12 L
12/06/2017	25,20 L

Fonte: Elaborada pelo autor.

Ao final de todos os dias foram recebidos relatórios diários sobre o consumo, conforme o esperado. Como exemplo tem-se o e-mail do último dia na Figura 31.

Figura 31 - E-mail com o relatório diário de consumo do último dia

**UORER** 20:31 (Há 3 horas)

para mim

Olá Rômulo!

Durante o último dia, desde 20h de ontem, você consumiu **37.80%** da sua meta diária.

**Parabéns!** Você está no caminho certo!

Consumo:

Intervalo	m <sup>3</sup>	L	R\$	Porcentagem da meta (%)
<b>Dia (Desde 20h de ontem)</b>	0.03	25.20	0.10	<b>37.80</b>
<b>Mês (Desde 07/06 as 23:58)</b>	0.10	96.90	0.37	<b>4.84</b>

Fonte: Elaborada pelo autor.

Dado que os resultados obtidos foram condizentes com a realidade pode-se dizer que o sistema funcionou conforme foi projetado.

## 6 CONCLUSÃO

A escolha da *framework Flask* para o desenvolvimento do código do lado do servidor foi adequada por proporcionar uma série de extensões que aceleraram o desenvolvimento do projeto ainda mantendo um grau de simplicidade, favorecendo a manutenção do código. De forma parecida o uso do Arduino UNO auxiliou o projeto pela simplicidade da linguagem de programação utilizada assim como uma grande comunidade, que igualmente ao *Flask*, forneceram apoio através de fóruns e repositórios de projetos.

Depois do desenvolvimento do protótipo, surgiram alguns problemas na sua instalação. Diferente do monitoramento de energia elétrica, a telemetria do consumo de água sofre de dois problemas não previstos no início do projeto: a ausência de um ponto de conexão com a rede elétrica e os perigos envolvendo circuitos eletrônicos em ambientes com presença de água, como exposição direta ou indireta a água e risco de choque elétrico. Tais problemas, se antevistos, teriam alterado algumas especificações do projeto como baixo consumo de energia, para o medidor funcionar por um longo período de tempo apenas com bateria, e o desenvolvimento de um protótipo menor e facilmente encapsulado, para prevenir com mais segurança o contato com a água.

Os dados enviados pelo medidor foram consistentes por apresentarem valores realistas de uso de água assim como proporcionarem uma visão do consumo em tempo real através do *website*. O conjunto medidor inteligente e aplicativo *web* funcionaram de acordo com o projetado por possibilitarem ver o consumo em tempo real através da interface *web* bem como o usuário ter recebido por e-mail relatórios de consumo ao final de cada dia.

Após o término deste trabalho pode-se afirmar que um sistema eficaz de monitoramento de consumo de água foi desenvolvido. Os resultados experimentais comprovam a efetividade do sistema em identificar o consumo e mandar as informações ao servidor, propiciando ao usuário uma visão detalhada dos gastos de parte da instalação hidráulica. O sistema desenvolvido já pode ser utilizado em contextos cuja vazão usualmente ultrapasse 1 L/min, como banheiros, empoderando indivíduos e grupos a reduzir o consumo de água através de informações em tempo real da utilização de água.

Por se tratar de um primeiro protótipo, algumas melhorias ainda devem ser implementadas para se obter um produto final. Na parte do medidor inteligente sugere-se para trabalhos futuros:

- a) Construção de medidor mais compacto, abandonando a placa do Arduino UNO;

- b) Mudança de protocolo para um mais enxuto e de menor latência. Apesar de simples e robusto o HTTP apresentou alguns empecilhos para o medidor inteligente: grande consumo de dados por ser um protocolo textual e voltado mais para navegadores *web* assim como latência elevada devido a necessidade de cada envio de dados abrir uma conexão TCP.;
- c) Estudo sobre a segurança dos dados enviados pela internet para evitar interceptação das informações sobre consumo;
- d) Desenvolvimento de outro modelo que seja sensível a pequenas vazões, possibilitando medir o consumo total de uma residência.

Já para o aplicativo *web* sugere-se implementar:

- a) Sistema de autenticação de usuários;
- b) Interface para manejear diversos medidores, caso o usuário possua mais de um;
- c) Interface para construir e analisar uma rede de medidores inteligentes, no contexto em que a instalação hidráulica seja extensa;
- d) Sistema de cadastramento de usuários agregados;
- e) Algoritmo para detecção de vazamentos.

## REFERÊNCIAS

**SUASSUNA, João. SEMI-ÁRIDO: proposta de convivência com a seca.** Recife, PE: Fundação Joaquim Nabuco, 2002. Disponível em: <[http://www.fundaj.gov.br/index.php?option=com\\_content&id=659&Itemid=376](http://www.fundaj.gov.br/index.php?option=com_content&id=659&Itemid=376)>. Acesso em: 11 de junho de 2017.

**IEEE. Towards a definition of the Internet of Things (IoT).** Itália, 2015. Disponível em: <[http://iot.ieee.org/images/files/pdf/IEEE\\_IoT\\_Towards\\_Definition\\_Internet\\_of\\_Things\\_Revision1\\_27MAY15.pdf](http://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf)>. Acesso em: 17 de maio de 2017.

**PFISTER, Cuno. Getting Started with the Internet of Things.** Sebastopol: Ed. O'Reilly Media, 2011.

**HERSENT, Olivier; BOSWARTHICK, David; ELLOUMI, Omar. The Internet of Things: Key Applications and Protocols.** Chichester: Ed. John Wiley & Sons Ltd, 2012.

**LIFEWIRE. Improve Your Understanding of Web Applications:** What Is a Web Application. Estados Unidos, 2016. Disponível em: <<https://www.lifewire.com/what-is-a-web-application-3486637>>. Acesso em: 17 de maio de 2017.

**MICRIUM. IoT for Embedded Systems:** The New Industrial Revolution. [S.I.], [201-?]. Disponível em: <<https://www.micrium.com/iot/introduction>>. Acesso em: 20 de maio de 2017.

**ELETTRONIC DESIGN. What's The Difference Between Bluetooth Low Energy and ANT.** [S.I.], 2012. Disponível em: <<http://www.electronicdesign.com/mobile/what-s-difference-between-bluetooth-low-energy-and-ant>>. Acesso em: 20 de maio de 2017.

**WIKIPEDIA. Z-Wave.** [S.I.], 2017. Disponível em: <<https://en.wikipedia.org/wiki/Z-Wave>>. Acesso em: 20 de maio de 2017.

**HAREENDRAN, T. K. Working with Water Flow Sensors & Arduino.** [S.I.], [2015?]. Disponível em: <<http://www.electroschematics.com/12145/working-with-water-flow-sensors-arduino>>. Acesso em: 17 de maio de 2017.

**SAIER. Datasheet HZ21WA.** [S.I.], [201-?]. Disponível em: <<http://www.microelectronicos.com/datasheets/UCTS0058.pdf>>. Acesso em: 17 de maio de 2017.

**ARDUINO. What is Arduino.** [S.I.], [201-?]. Disponível em: <<https://www.arduino.cc/en/Guide/Introduction>>. Acesso em: 23 de maio de 2017.

**ARDUINO. Arduino Uno & Genuino Uno.** [S.I.], [201-?]. Disponível em: <<https://www.arduino.cc/en/main/arduinoBoardUno>>. Acesso em: 23 de maio de 2017.

**MICROCHIP. ATmega328P.** Estados Unidos, 2016. Disponível em: <<http://www.microchip.com/wwwproducts/en/ATmega328P>>. Acesso em: 20 de maio de 2017.

**WEB-ENGINEERING. Javascript-based IoT/WoT Development with the ESP8266 and**

**the Raspberry Pi.** [S.I.], 2016. Disponível em: <<http://web-engineering.info/node/65>>. Acesso em: 10 de junho de 2017.

**DEVELOPERSHOME. 14. Introduction to AT Commands.** [S.I.], [200-?]. Disponível em: <<http://www.developershome.com/sms/atCommandsIntro.asp>>. Acesso em: 10 de junho de 2017.

**ESPRESSIF. ESP8266 Technical Reference.** Versão 1.3. [S.I.], 2017. Disponível em: <[https://espresif.com/sites/default/files/documentation/esp8266-technical\\_reference\\_en.pdf](https://espresif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf)>. Acesso em: 14 de julho de 2017.

**ESPRESSIF. ESP8266 AT Instruction Set.** Versão 2.1.0. [S.I.], 2017. Disponível em: <[https://www.espressif.com/sites/default/files/documentation/4a-esp8266\\_at\\_instruction\\_set\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf)>. Acesso em: 14 de julho de 2017.

**BANGGOOD. Geekcreit® ESP8266 Serial Wi-Fi Wireless ESP-01 Adapter Module 3.3V 5V Compatible For Arduino.** [S.I.], [2016?]. Disponível em: <[https://www.banggood.com/ESP8266-Serial-Wi-Fi-Wireless-ESP-01-Adapter-Module-3\\_3V-5V-Compatible-For-Arduino-p-1047322.html?rmmds=myorder](https://www.banggood.com/ESP8266-Serial-Wi-Fi-Wireless-ESP-01-Adapter-Module-3_3V-5V-Compatible-For-Arduino-p-1047322.html?rmmds=myorder)>. Acesso em: 10 de junho de 2017.

**ALLDATASHEET. 1N5819 Datasheet (PDF) – Motorola, Inc.** [S.I.], [200-?]. Disponível em: <<http://pdf1.alldatasheet.com/datasheet-pdf/view/2818/MOTOROLA/1N5819.html>>. Acesso em: 26 de junho de 2017.

GRINBERG, Miguel. **FLASK Web Development: Developing web applications with python.** Sebastopol: Ed. O'Reilly Media, 2014.

SOLEM, Ask. **Celery: Distributed Task Queue.** [S.I.], 2017. Disponível em: <<http://www.celeryproject.org/>>. Acesso em: 14 de julho de 2017.

GRINBERG, Miguel. **Using Celery With Flask.** [Portland], 2015. Disponível em: <<https://blog.miguelgrinberg.com/post/using-celery-with-flask>>. Acesso em: 14 de julho de 2017.

## APÊNDICE A – FIRMWARE DO MEDIDOR INTELIGENTE

```

#include "SoftwareSerial.h"
String ssid =<NOME-DA-REDE-WIFI>;
String password =<SENHA-DA-REDE-WIFI>;
String server =<URL-DO-SERVIDOR>;
String uri =<URI-DO-SERVIDOR>;

SoftwareSerial esp(6, 7); // RX, TX
int flowPin = 2;           //pino de interrupção do Arduino UNO

// Do tipo volatile para garantir a atualização correta da variável durante interrupções
volatile int64_t contaPulsos = 0LL;
// 'LL' para indicar que é long long, ou inteiro de 64 bit
// vai de -2^63 a 2^63-1 = 9223372036854775807, o que equivale ao consumo acumulado de
// aproximadamente 37.500 km³ de água
String data; // string que conterá a medição no formato JSON

// reiniciar ESP8266
void reset() {
    esp.println("AT+RST");
    delay(50);
    if(esp.find("OK")) {
        Serial.println("Módulo Wi-Fi reiniciado com sucesso!");
    } else {
        Serial.println("Não foi possível resetar o módulo Wi-Fi.");
        delay(1000);
        reset();
    }
}

// conectar ESP8266 ao Wi-Fi
void connectWifi() {
    String cmd = "AT+CWJAP=\"" + ssid + "\",\"" + password + "\"";
    esp.println(cmd);
    delay(3000);
    if(esp.find("OK")) {
        Serial.println("Conectado ao Wi-Fi!");
    } else {
        Serial.println("Não foi possível conectar ao Wi-Fi.");
        delay(1000);
        connectWifi();
    }
}

// Conversão de int64_t para string
String int64_to_string(int64_t valor) {
    String resultadoParcial;
    String resultado;
    int64_t q = valor;
    do {
        resultadoParcial += "0123456789"[q%10];
        q /= 10;
    } while(q > 0);

    // fazendo o inverso da string resultadoParcial
    for(int i = resultadoParcial.length()-1; i >= 0 ; i--) {
        resultado += resultadoParcial[i];
    }
}

```

```

    return resultado;
}

// post HTTP
void httppost (int64_t nPulos) {
    data = "{\"valor\":\"" + int64_to_string(nPulos) + "\"}"; // dados enviados no formato JSON
    esp.println("AT+CIPSTART=\"TCP\",\"" + server + "\",5000"); // inicia conexão TCP

    if(esp.find("OK")) {
        Serial.println("Conexão TCP pronta.");
    }

    String postRequest =
    "POST " + uri + " HTTP/1.1\r\n" +
    "Accept: application/json, */*\r\n" +
    "Connection: close\r\n" +
    "Content-Length: " + data.length() + "\r\n" +
    "Content-Type: application/json\r\n" +
    "Host: " + server + "\r\n" +
    "\r\n" +
    data;

    String sendCmd = "AT+CIPSEND="; //determine the number of caracters to be sent.
    esp.print(sendCmd);
    esp.println(postRequest.length());

    while(esp.available() > 0){esp.read();}

    if(esp.find(">")) {
        Serial.println("Sending..");
        esp.print(postRequest);
        if(esp.find("SEND OK")) {
            Serial.println("Packet sent");
            // close the connection
            esp.println("AT+CIPCLOSE");
        }
    }
}

// Função que conta os pulsos do sensor de fluxo
void Flow(){contaPulos++;}

void setup() {
    pinMode(flowPin, INPUT);
    attachInterrupt(digitalPinToInterrupt(flowPin), Flow, RISING);
    interrupts();

    esp.begin(9600);
    Serial.begin(9600);

    reset();
    connectWifi();
}

void loop () {
    delay (500);
    httppost(contaPulos);
}

```

## APÊNDICE B – CÓDIGO DO LADO DO SERVIDOR

```

#!/usr/bin/python
# -*- coding: utf-8 -*-

# Código inicial em único script para facilitar desenvolvimento
import os, sys
reload(sys)
sys.setdefaultencoding('utf-8')

import os
from flask import Flask, render_template, session, redirect, url_for, flash
from flask_script import Manager, Shell
from flask_wtf import Form
from wtforms import FloatField, SubmitField
from flask_bootstrap import Bootstrap
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime, timedelta
from flask_migrate import Migrate, MigrateCommand
from flask import Blueprint
from flask import jsonify, request
from flask_socketio import SocketIO, emit, join_room, leave_room, close_room, rooms, disconnect
from sqlalchemy import extract
from flask_moment import Moment
from flask_mail import Mail, Message

from threading import Thread
from celery import Celery
from celery.schedules import crontab

cte = 1.519064256 # mL/pulso [sensor de vazão]
PM = 3.85      # R$/m³ [Preço médio do m³ de água]

basedir = os.path.abspath(os.path.dirname(__file__))

async_mode = None

app = Flask(__name__)
api = Blueprint('api', __name__)
app.config['SECRET_KEY'] = os.environ.get('SECRET_KEY') # (usado para proteger webforms)
socketio = SocketIO(app, async_mode=async_mode)
thread = None

# Configurando Database
app.config['SQLALCHEMY_DATABASE_URI'] = \
    'sqlite:/// + os.path.join(basedir, 'data.sqlite') # Usando db SQLite para um desenvolvimento simples
app.config['SQLALCHEMY_COMMIT_ON_TEARDOWN'] = True    # Permite commits automáticos das
mudanças do db após
                                # o final de cada request
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False  # ao iniciar o shell foi pedido para
desativá-lo se não for usado

# Configurando e-mail
app.config['MAIL_SERVER'] = 'smtp.googlemail.com'
app.config['MAIL_PORT'] = 587

```

```

app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = os.environ.get('MAIL_USERNAME')
app.config['MAIL_PASSWORD'] = os.environ.get('MAIL_PASSWORD')
app.config['MAIL SUBJECT PREFIX'] = '[UORER] '
app.config['MAIL_SENDER'] = 'UORER <uorer.adm@gmail.com>'
app.config['MAIL_ADMIN'] = os.environ.get('MAIL_ADMIN')

# Configurando o Celery
app.config['CELERY_BROKER_URL'] = 'redis://localhost:6379/0'
app.config['result_backend'] = 'redis://localhost:6379/0'

celery = Celery(app.name, broker=app.config['CELERY_BROKER_URL'])
celery.conf.update(app.config)

db = SQLAlchemy(app)
db.text_factory = unicode
manager = Manager(app)
bootstrap = Bootstrap(app)
migrate = Migrate(app, db)
manager.add_command('db', MigrateCommand)
moment = Moment(app)
mail = Mail(app)

def make_shell_context():
    return dict(app=app, db=db, Usuario=Usuario, Medidor=Medidor, Meta=Meta, Medicao=Medicao,
               meta=Meta.query.first(),
               eu=Usuario.query.first())

manager.add_command("shell", Shell(make_context=make_shell_context))

@celery.on_after_configure.connect
def setup_periodic_tasks(sender, **kwargs):
    # Executa 'relatorioDiario' todo dia as 20h (UTC)
    sender.add_periodic_task(
        crontab(hour=20, minute=0),
        relatorioDiario.s(),
    )

    # Executa 'desnotificarDiario' todo dia as 00h UTC
    sender.add_periodic_task(
        crontab(hour=0, minute=0),
        desnotificarDiario.s(),
    )

    # Executa 'desnotificarMensal' todo dia 10 do mês as 0h (UTC)
    sender.add_periodic_task(
        crontab(0, 0, day_of_month='10'),
        desnotificarMensal.s(),
    )

@celery.task
def test(arg):
    print(arg)

@celery.task
def desnotificarDiario():
    with app.app_context():

```

```

usuarios = Usuario.query.all()
for usuario in usuarios:
    meta.usuario.medidores.first().meta
    # Limpando notificação da meta diária
    meta.desnotificar("dia", "100%")
    meta.desnotificar("dia", "80%")

@celery.task
def desnotificarMensal():
    with app.app_context():
        usuarios = Usuario.query.all()
        for usuario in usuarios:
            meta = usuario.medidores.first().meta
            # Limpando a notificação da meta mensal
            meta.desnotificar("mês", "100%")
            meta.desnotificar("mês", "80%")

@celery.task
def relatorioDiario():
    with app.app_context():
        usuarios = Usuario.query.all()
        for usuario in usuarios:
            meta = usuario.medidores.first().meta

            agora = datetime.utcnow()
            inicioMes = meta.inicio

            while inicioMes < agora:
                inicioMes += meta.intervalo
                inicioMes -= meta.intervalo

            inicioMetaMensal = "{:02d}/{:02d} as {:02d}:{:02d}".format(inicioMes.day, inicioMes.month,
            inicioMes.hour, inicioMes.minute)

            """
            CONSUMO ACUMULADO NO DIA E NO MÊS
            """
            ultimaMedicao = Medicao.query.order_by(Medicao.id.desc()).first() # pegando o último valor

            # última medição do mês passado
            ultimaMedicaoMesPassado = \
                Medicao.query.filter(Medicao.dataHora <= inicioMes).order_by(Medicao.id.desc()).first()
            ultimaMedicaoMesPassado = 0 if(ultimaMedicaoMesPassado is None) else
            ultimaMedicaoMesPassado.valor

            consumoMes = (ultimaMedicao.valor - ultimaMedicaoMesPassado)*cte/1000000 # m³

            # consumo do dia em m³
            ontem = datetime(agora.year, agora.month, agora.day-1, inicioMes.hour, inicioMes.minute)
            ultimaMedicaoOntem = \
                Medicao.query.filter(Medicao.dataHora <= ontem).order_by(Medicao.id.desc()).first()

            ultimaMedicaoOntem = 0 if(ultimaMedicaoOntem is None) else ultimaMedicaoOntem.valor # caso não
            exista consumo nos dias anteriores

            consumoDia = (ultimaMedicao.valor - ultimaMedicaoOntem)*cte/1000000 # m³, atenção porque
            ultimaMedicaoOntem já é um float,
            # não um objeto Medicao

```

```

# consumos em m³, L e R$
consumoMes = {
    "m³": consumoMes
    , "L": consumoMes*1000
    , "R$": consumoMes*PM
}
consumoDia = {
    "m³": consumoDia
    , "L": consumoDia*1000
    , "R$": consumoDia*PM
}

consumoMes["%"] = consumoMes[meta.unidadeDoValor.encode('utf-8')]/meta.valor*100
consumoDia["%"] = consumoDia[meta.unidadeDoValor.encode('utf-8')]/(meta.valor/30)*100 # supondo
mês com 30 dias

sendAsyncEmail.delay({ #(para, assunto, template, **kwargs):
    "para": usuario.email
    , "assunto": 'Relatório diário'
    , "template": 'email/relatorioDiario'
    , "kwargs": {
        "usuario": usuario.nome
        , "horaMeta": meta.inicio.hour - 3 # conversão temporária para o horário brasileiro
        , "inicioMetaMensal": inicioMetaMensal
        , "consumo": {
            "dia": consumoDia
            , "mês": consumoMes
        }
    }
})

@celery.task
def sendAsyncEmail(parametros): #(para, assunto, template, **kwargs):
    with app.app_context():
        msg = Message(app.config['MAIL SUBJECT PREFIX'] + parametros["assunto"],
                      sender=app.config['MAIL SENDER'], recipients=[parametros["para"]])
        msg.body = render_template(parametros["template"] + '.txt', **parametros["kwargs"])
        msg.html = render_template(parametros["template"] + '.html', **parametros["kwargs"])
        mail.send(msg)

def analisarMetaNotificar(meta):
    if meta.foiNotificado("dia", "100%") and meta.foiNotificado("mês", "100%"):
        pass # fazer nada, o usuário já foi notificado pela ultrapassagem dos 100% diário e mensal
    else:
        """ CONSUMO ACUMULADO NO DIA E NO MÊS """
        # Consumo do mês em m³
        ultimaMedicao = Medicao.query.order_by(Medicao.id.desc()).first()
        consumoMes = ultimaMedicao.valor*cte/1000000 # m³

        # Consumo do dia em m³
        # Primeiro pega-se a última medição do dia anterior. No caso utilizamos '<=' para o caso que o medidor não
        tenha consumo no
        # dia anterior
        agora = datetime.utcnow()
        ultimaMedicaoOntem = \
            Medicao.query.filter(Medicao.dataHora <= datetime(agora.year, agora.month,
            agora.day)).order_by(Medicao.id.desc()).first()
        consumoDia = ultimaMedicaoOntem.valor if (ultimaMedicaoOntem is not None) else 0 # caso não exista

```

```

consumo nos dias anteriores
    consumoDia = consumoMes - consumoDia*cte/1000000 # m3

    # consumos em m3, L e R$
    consumoMes = {
        "m333] / (meta.valor/30)*100 # supondo mês com 30 dias
    , "mês": consumoDiaMes["mês"]["m3] / meta.valor*100
}

for key in consumidoPorcento:
    # Análise dos 100%
    if meta.foiNotificado(key, "100%"):
        pass # fazer nada, o usuário já foi notificado pela passagem de 100% da meta 'key' (diária ou mensal)
    elif consumidoPorcento[key] >= 100:
        meta.notificar(key, "100%") # alteração do db e e-mail

    # Análise dos 80% (note que só será enviado um único e-mail para a meta diária e outro para meta
    mensal)
    elif meta.foiNotificado(key, "80%"):
        pass # faz nada, o usuário já foi notificado pela ultrapassagem dos 80%
    elif consumidoPorcento[key] >= 80:
        meta.notificar(key, "80%") # alteração do db e e-mail

#####
## Modelos do DB ##
#####

### Definição de Permissões dos Usuários #####
# Permissões são definidas com um byte, tendo um dos bits igual a 1 e
# os outros iguais a zero. O nível da permissão é baseado na posição do
# seu bit igual a 1, e aumenta da direita para a esquerda.
# A permissão de "ADMINISTRAR" é especial e tem todos os bits iguais a 1.
class Permissao:
    VISUALIZAR = 0x01
    CADASTRAR = 0x02
    ADMINISTRAR = 0xff

class Cargo(db.Model): # Tipos de usuários no sistema
    __tablename__ = 'cargos'

```

```

id = db.Column(db.Integer, primary_key=True)
nome = db.Column(db.String(30), index=True, unique=True, nullable=False)

# Permissões dos usuários do cargo (interpretado como um byte, em que cada
# bit representa uma permissão, e o estado do bit indica se o usuário tem (1) ou
# não (0) tal permissão)
permissoes = db.Column(db.Integer, nullable=False)

# Relação de usuários que possuem o cargo
usuarios = db.relationship('Usuario', backref='cargo', lazy='dynamic')

#### Métodos ####

# Adicionar os cargos no banco de dados
@staticmethod
def criar_cargos():
    # Definição dos cargos e suas permissões
    cargos = {
        'Agregado': Permissao.VISUALIZAR
        , 'Proprietário': Permissao.VISUALIZAR
        , 'Desenvolvedor': Permissao.ADMINISTRAR
        , 'Administrador': Permissao.ADMINISTRAR
    }

    # Criação dos cargos
    for nome_cargo in cargos:
        cargo = Cargo.query.filter_by(nome=nome_cargo).first()

        if cargo is None:
            cargo = Cargo(nome=nome_cargo)
            cargo.permissoes = cargos[nome_cargo]
            db.session.add(cargo)

    # Salvando no banco de dados
    db.session.commit()

# Representação no shell
def __repr__(self):
    return '<Cargo: %s>' % self.nome

# Representação na interface
def __str__(self):
    return self.nome


# Tabela de associação de usuários proprietários e agregados
hierarquia = db.Table('hierarquia',
    db.Column('proprietarioID', db.Integer, db.ForeignKey('usuarios.id')),
    db.Column('agregadoID', db.Integer, db.ForeignKey('usuarios.id'))
)

class Usuario(db.Model):
    __tablename__ = 'usuarios' # nome da tabela no banco de dados
    id = db.Column(db.Integer, primary_key=True)
    nome = db.Column(db.String(64), index=True)
    endereco = db.Column(db.String(128))
    email = db.Column(db.String(64), index=True, unique=True, nullable=False)
    # Hash da senha (a senha original não é armazenada)
    senhaHash = db.Column(db.String(128))

```

```

# Indica se o usuário confirmou o email
confirmado = db.Column(db.Boolean, default=False)

# Relacionamento de proprietários e agregados da tabela 'usuarios'
# Usuários agregados são aqueles em que um proprietário concede o privilégio de visualização de seus
# medidores
agregados = db.relationship('Usuario',
                            secondary=hierarquia,
                            primaryjoin=(hierarquia.c.proprietarioID == id),
                            secondaryjoin=(hierarquia.c.agregadoID == id),
                            backref=db.backref('proprietarios', lazy='dynamic'),
                            lazy='dynamic')
# proprietarios.all() = [todos os proprietarios que tem como agregado o usuário atual]

# Relação com a tabela 'cargos'
cargoID = db.Column(db.Integer, db.ForeignKey('cargos.id'))

# Relação de medidores de um usuário
medidores = db.relationship('Medidor', backref='usuario', lazy='dynamic')

#### Métodos ####

# Criar o primeiro administrador, caso ainda não haja um
@staticmethod
def criar_administrador():
    if not Usuario.query.join(Cargo).filter(Cargo.nome=='Administrador').first():
        # Definição dos dados (alguns obtidos em variáveis de ambiente)
        administrador = Usuario(email=current_app.config['ADMIN_EMAIL'])
        administrador.nome = 'Administrador'
        administrador.senha = current_app.config['ADMIN_SENHA']
        administrador.verificado = True
        administrador.confirmado = True
        administrador.cargo = Cargo.query.filter_by(nome='Administrador').first()

        # Salvando no banco de dados
        db.session.add(administrador)
        db.session.commit()

    # Representação no shell
    def __repr__(self):
        return '<Usuário: %s>' % self.nome #[%s]>' % (self.nome, self.cargo.nome)

    # Representação na interface
    def __str__(self):
        return self.nome

class ModeloMedidor(db.Model):
    __tablename__ = 'modelos_dos_medidores'
    id = db.Column(db.Integer, primary_key=True)
    nome = db.Column(db.String(32), index=True)

    # Relação de medidores de um modelo
    medidores = db.relationship('Medidor', backref='modelo', lazy='dynamic')

#### Métodos ####

```

```

# Adicionar os modelos ao banco de dados
@staticmethod
def criar_modelos():
    # Definição dos modelos e suas permissões
    modelos = [
        u'1.0',
        ,u'1.1',
        ,u'2.0'
    ]

    # Criação dos modelos
    for nome_modelo in modelos:
        modelo = ModeloMedidor.query.filter_by(nome=nome_modelo).first()
        if modelo is None:
            modelo = ModeloMedidor(nome=nome_modelo)
            db.session.add(modelo)
    # Salvando no banco de dados
    db.session.commit()

    # Representação no shell
    def __repr__(self):
        return '<Modelo de medidor: %s>' % self.nome

    # Representação na interface
    def __str__(self):
        return self.nome

# Tabela de associação de medidores pais e filhos
genealogia = db.Table('genealogia',
    db.Column('paiID', db.Integer, db.ForeignKey('medidores.id')),
    db.Column('filhoID', db.Integer, db.ForeignKey('medidores.id'))
)

class Medidor(db.Model):
    tablename = 'medidores'
    id = db.Column(db.Integer, primary_key=True)
    # Nome para facilitar a identificação do medidor, pode ser o complemento da frase "Medidor do(a) ..."
    nome = db.Column(db.String(64), index=True)
    # Preço médio do m³ de água em R$
    precoMedio = db.Column(db.Float)
    # Constante do medidor que informa quantos m³/pulso ele mede
    cte = db.Column(db.Float)
    # Um endereço para o medidor, já que este pode ser diferente do pertencente ao proprietário
    endereco = db.Column(db.String(128))

    # Relacionamento de pais e filhos na tabela 'medidores'
    filhos = db.relationship('Medidor',
        secondary=genealogia,
        primaryjoin=(genealogia.c.paiID == id),
        secondaryjoin=(genealogia.c.filhoID == id),
        backref=db.backref('pais', lazy='dynamic'),
        lazy='dynamic')
    # pais.all() = [todos os medidores acima do medidor atual]

    # Relação com a tabela 'usuarios'
    usuarioID = db.Column(db.Integer, db.ForeignKey('usuarios.id'))
    # Relação com a tabela 'modelos_dos_medidores'
    modelo_do_medidorID = db.Column(db.Integer, db.ForeignKey('modelos_dos_medidores.id'))
    # Relação com a tabela 'metas'

```

```

metaID = db.Column(db.Integer, db.ForeignKey('metas.id'))

# Relação com as tabela 'medicoes'
medicoes = db.relationship('Medicao', backref='medidor', lazy='dynamic')

### MÉTODOS ###

# Representação no shell
def __repr__(self):
    return '<Medidor ID: %s>' % self.nome

# Representação na interface
def __str__(self):
    return self.nome

class Meta(db.Model):
    __tablename__ = 'metas'
    id = db.Column(db.Integer, primary_key=True)
    # Descrição da meta
    descricao = db.Column(db.String(64), index=True)
    # Valor numérico da meta
    valor = db.Column(db.Float)
    # Unidade de medida da meta
    unidadeDoValor = db.Column(db.String(20))
    # Início da meta, por exemplo: 10/03/2017
    inicio = db.Column(db.DateTime, index=True)
    # Intervalo de tempo para atingir a meta (Dia, Semana, Mês, Ano)
    intervalo = db.Column(db.Interval)
    # Tempo entre reatórios por e-mail ao usuário sobre a meta
    notificacoes = db.Column(db.Interval)

    # Guardando num número binário se um alerta já foi emitido, sendo os bits da seguinte ordem:
    # [notificadoDia100%, notificadoMes100%, notificadoDia80%, notificadoMes80%]
    notificado = db.Column(db.Integer, default=0b0000)

# Relação medidor de uma meta
medidores = db.relationship('Medidor', backref='meta', lazy='dynamic')

"""

MÉTODOS
"""

def foiNotificado(self, intervalo, porcento):
    switch = {
        "100%": {
            "dia": self.notificado & 0b1000 == 0b1000
            , "mês": self.notificado & 0b0100 == 0b0100
        }
        , "80%": {
            "dia": self.notificado & 0b0010 == 0b0010
            , "mês": self.notificado & 0b0001 == 0b0001
        }
    }
    return switch[porcento][intervalo]

def notificar(self, intervalo, porcento):
    if not self.foiNotificado(intervalo, porcento): # segurança para não mandar o e-mail mais de uma vez
        # Mandando o e-mail
        assunto = {

```

```

    "100%": {
        "dia": "Ultrapassagem de meta diária!"
        ,"mês": "Ultrapassagem de meta mensal!"
    }
    , "80%": {
        "dia": "80% da meta diária atingida"
        , "mês": "80% da meta mensal atingida"
    }
}

sendAsyncEmail.delay({
    "para": self.medidores.first().usuario.email
    , "assunto": assunto[porcento][intervalo]
    , "template": 'email/alertaMeta'
    , "kwargs": {
        "intervaloMeta": intervalo
        , "porcentagem": porcento
    }
})

# Atualizando o database
switch = {
    "100%": {
        "dia": self.notificado | 0b1000
        , "mês": self.notificado | 0b0100
    }
    , "80%": {
        "dia": self.notificado | 0b0010
        , "mês": self.notificado | 0b0001
    }
}

self.notificado = switch[porcento][intervalo]
db.session.add(self)
db.session.commit()

def desnotificar(self, intervalo, porcento):
    switch = {
        "100%": {
            "dia": self.notificado & 0b0111
            , "mês": self.notificado & 0b1011
        }
        , "80%": {
            "dia": self.notificado & 0b1101
            , "mês": self.notificado & 0b1110
        }
    }
    self.notificado = switch[porcento][intervalo]
    db.session.add(self)
    db.session.commit()

# Representação no shell
def __repr__(self):
    return '<Meta: %s>' % self.descricao

# Representação na interface
def __str__(self):

```

```

    return self.descricao

class Medicao(db.Model):
    __tablename__ = 'medicoes' # nome da tabela no banco de dados
    id = db.Column(db.Integer, primary_key = True)
    # Número de pulsos, enviado pelo medidor
    valor = db.Column(db.Float)
    # Data e hora que o servidor recebeu a medição
    dataHora = db.Column(db.DateTime, index = True, default=datetime.utcnow)

    # Relação com a tabela 'medidores'
    medidorID = db.Column(db.Integer, db.ForeignKey('medidores.id'))

    #### Métodos ####

    # Converter uma medição em JSON
    def to_json(self):
        jsonMedicao = {
            'valor': self.valor,
            'dataHora': self.dataHora
        }
        return jsonMedicao

    # Converter um JSON numa Medicao
    @staticmethod
    def from_json(jsonMedicao):
        valorDaMedicao = jsonMedicao.get('valor')
        if valorDaMedicao is None:
            return 'bug'
            # raise ValidationError('dado enviado sem valor atribuído')
        return Medicao(valor = valorDaMedicao)

    # Representação no shell
    def __repr__(self):
        return '<Medição: %s>' % self.id

    # Representação na interface
    def __str__(self):
        return self.id

    #####
    # Formulário de aquisição de dados
    class DataForm(Form):
        dado = FloatField('Insira um float:')
        submit = SubmitField('Enviar')

    #####
    # View functions do webapp
    @app.route('/', methods = ['GET', 'POST'])
    def index():

        minhaMeta = Meta.query.first()

        minhaMeta = {
            "valor": {
                "m³": minhaMeta.valor

```

```

        ,"L": minhaMeta.valor*1000
        ,"R$": minhaMeta.valor*PM
    }
    , "unidadeDoValor": minhaMeta.unidadeDoValor
    , "inicio": minhaMeta.inicio.isoformat()+'Z'
    , "fim": (minhaMeta.inicio + minhaMeta.intervalo).isoformat()+'Z'
}

eu = Usuario.query.first()
eu = {
    "nome": eu.nome
    , "email": eu.email
    , "senha": eu.senhaHash
}

# 21 dados para poder calcular 20 últimas vazões
dados_ultimos21 = Medicao.query.all()[-21:]
vazao_ultimos20 = []
dataHora_ultimos20 = []
tempTxt = ""
temp = 0
for i in range(1,len(dados_ultimos21)):
    """
        Os dados em formato datetime serão utilizados no lado do cliente em conjunto com a biblioteca moment.js
        para oferecer a conversão da data e hora de acordo com a localização e configuração do usuário. O que vai
        acontecer no javascript do cliente é:
        moment("2012-12-31T23:55:13Z").format('LLLL');
        Pra isso tem que ser enviado no lugar do objeto datetime uma string usando isoformat(), como:
        obj.isoformat();
        que coloca um 'T' entre a data e a hora e depois adicionar um 'Z' no final da string pro moment.js
        reconhecer a parada
    """
    tempTxt = dados_ultimos21[i].dataHora.isoformat()+'Z'
    dataHora_ultimos20.append(tempTxt)
    """
        (60 s/min)*(1/1000 L/mL)*(cte mL/pulsos)*(intervaloDeConsumo pulsos)/(intervaloDeTempo s)
        = 0.06*cte*intervaloDeConsumo/intervaloDeTempo L/min
    """
    temp = (0.06*cte)*(dados_ultimos21[i].valor - dados_ultimos21[i-1].valor)/\
            (dados_ultimos21[i].dataHora - dados_ultimos21[i-1].dataHora).total_seconds() # L/min
    vazao_ultimos20.append(temp)

    """
        CONSUMO ACUMULADO NO DIA E NO MÊS
    """
    # consumo do mês em m³
    dado_ultimo = dados_ultimos21[-1]
    consumoMes = dado_ultimo.valor*cte/1000000 # m³
    # consumo do dia em m³
    # Primeiro pega-se a última medição do dia anterior. No caso utilizamos '<=' para o caso que o medidor não
    tenha consumo no
    # dia anterior
    agora = datetime.utcnow()
    ultimaMedicaoOntem = \
        Medicao.query.filter(Medicao.dataHora <= datetime(agora.year, agora.month,
        agora.day)).order_by(Medicao.id.desc()).first()
    consumoDia = ultimaMedicaoOntem.valor if (ultimaMedicaoOntem is not None) else 0 # caso não exista
    consumo nos dias anteriores
    consumoDia = consumoMes - consumoDia*cte/1000000 # m³

    # consumos em m³, L e R$
    consumoMes = {

```

```

        "m³": consumoMes
        ,"L": consumoMes*1000
        ,"R$": consumoMes*PM
    }
consumoDia = {
    "m³": consumoDia
    ,"L": consumoDia*1000
    ,"R$": consumoDia*PM
}

# dicionário final
consumoDiaMes = {
    "dia": consumoDia
    , "mês": consumoMes
}

consumoAcumuladoTotal = {
    "valor": Medicao.query.order_by(Medicao.id.desc()).first().valor # pegando o último valor
    , "dataHora": Medicao.query.first().dataHora.isoformat()+'Z'
}
consumoAcumuladoTotal['valor'] = cte*consumoAcumuladoTotal['valor']/1000000 # m³

historico_1mes = {
    "valor": {
        "m³": []
        , "L": []
        , "R$": []
    }
    , "dataHora": []
}
# supondo mês com 31 dias
temp = []
for i in range(32,0,-1): #[32, 30, ..., 2, 1]
    # última medição do dia i-ésimo dia anterior
    temp2 = Medicao.query.filter(extract('day', Medicao.dataHora) == datetime.utcnow().day-i).order_by(Medicao.id.desc()).first()
    if temp2 is not None:
        temp.append(temp2)
    if len(temp) > 1:
        consumoDoDia = (temp[-1].valor - temp[-2].valor)*cte/1000000 # m³
        historico_1mes["valor"]["m³"].append(consumoDoDia)
        historico_1mes["valor"]["L"].append(consumoDoDia*1000)
        historico_1mes["valor"]["R$"].append(consumoDoDia*PM)
        # Formato de dataHora para a biblioteca plotly.js
        historico_1mes["dataHora"].append("%d-%d-%d" %(temp[-1].dataHora.year, temp[-1].dataHora.month, temp[-1].dataHora.day))

return render_template('painelDeControle.html', async_mode=socketio.async_mode, cte=cte, PM=PM,
                      vazao_ultimos20=vazao_ultimos20, dataHora_ultimos20=dataHora_ultimos20,
                      consumoDiaMes=consumoDiaMes, minhaMeta=minhaMeta, eu=eu,
                      consumoAcumuladoTotal=consumoAcumuladoTotal, historico_1mes=historico_1mes)

# Página para enviar dados, simulando o trabalho do smartmeter
@app.route('/enviar', methods = ['GET', 'POST'])
def enviar():
    form = DataForm()
    if form.validate_on_submit():
        flash('Dados enviados.')

```

```

valor = form.dado.data
form.dado.data = ""
novaMedicao = Medicao(valor = valor)
db.session.add(novaMedicao)
db.session.commit()

minhaMeta = Meta.query.first()
analisarMetaNotificar(minhaMeta)

return redirect(url_for('enviar'))
return render_template('enviar.html', form = form)

# Construindo a API dentro da aplicação
@app.route('/api/dados/', methods = ['POST'])
def postDado():
    medicao = Medicao.from_json(request.json)
    db.session.add(medicao)
    db.session.commit()

    minhaMeta = Meta.query.first()
    analisarMetaNotificar(minhaMeta)

    return jsonify(medicao.to_json()), 201

@app.errorhandler(404)
def page_not_found(e):
    if request.accept_mimetypes.accept_json and not request.accept_mimetypes.accept_html:
        response = jsonify({'erro': 'não encontrado'})
        response.status_code = 404
        return response
    return render_template('404.html'), 404

@app.errorhandler(500)
def internal_server_error(e):
    if request.accept_mimetypes.accept_json and not request.accept_mimetypes.accept_html:
        response = jsonify({'erro': 'problemas no servidor'})
        response.status_code = 500
        return response
    return render_template('500.html'), 500

#####
# SOCKET IO #
def background_thread():
    with app.app_context(): # usando app_context para poder acessar o db
        ultimaMedicao = Medicao.query.order_by(Medicao.id.desc()).first()
        while True:
            socketio.sleep(1)
            if ultimaMedicao != Medicao.query.order_by(Medicao.id.desc()).first():
                [penultima, ultimaMedicao] = Medicao.query.all()[-2:]
                vazaoAtual = (0.06*cte)*(ultimaMedicao.valor - penultima.valor) /
                    (ultimaMedicao.dataHora - penultima.dataHora).total_seconds() # L/min
                """
                CONSUMO ACUMULADO NO DIA E NO MÊS
                """
                # consumo do mês em m³
                consumoMes = ultimaMedicao.valor*cte/1000000 # m³

```

```

# consumo do dia em m³
# Primeiro pega-se a última medição do dia anterior. No caso utilizamos '<=' para o caso que o
medidor não tenha consumo no
# dia anterior
agora = datetime.utcnow()
ultimaMedicaoOntem = \
    Medicao.query.filter(Medicao.dataHora <= datetime(agora.year, agora.month,
agora.day)).order_by(Medicao.id.desc()).first()
consumoDia = ultimaMedicaoOntem.valor if (ultimaMedicaoOntem is not None) else 0 # caso não
exista consumo nos dias anteriores
consumoDia = consumoMes - consumoDia*cte/1000000 # m³

# consumos em m³, L e R$
consumoMes = {
    "m³": consumoMes
    , "L": consumoMes*1000
    , "R$": consumoMes*PM
}
consumoDia = {
    "m³": consumoDia
    , "L": consumoDia*1000
    , "R$": consumoDia*PM
}

# dicionário final
consumoDiaMes = {
    "dia": consumoDia
    , "mês": consumoMes
}

socketio.emit('my_response'
    ,{'vazaoAtual': vazaoAtual, 'dataHora': ultimaMedicao.dataHora.isoformat()+'Z',
'consumoDiaMes': consumoDiaMes}
    ,namespace='/test')




@socketio.on('disconnect_request', namespace='/test')
def disconnect_request():
    session['receive_count'] = session.get('receive_count', 0) + 1
    disconnect()

@socketio.on('connect', namespace='/test')
def test_connect():
    global thread
    if thread is None:
        thread = socketio.start_background_task(target=background_thread)
    emit('statusConexao', {'data': 'Connectado ao servidor'})

@socketio.on('disconnect', namespace='/test')
def test_disconnect():
    print('Client disconnected', request.sid)

#####
if __name__ == '__main__':
    socketio.run(app, debug=True, host='0.0.0.0')
    #manager.run()

```