

INSTITUTO FEDERAL
SANTA CATARINA

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA
CAMPUS SÃO JOSÉ
ENGENHARIA DE TELECOMUNICAÇÕES

Relatório Final Controles de cargas apartir do kWh da unidade consumidora

Equipe: Lucas Thiesen e Mário Allan Lehmkuhl de Abreu

Matéria: Projeto Integrador 3

Professores: Diego Medeiros

Fevereiro
2019

1 Proposta

O objetivo do projeto é obter um controle de cargas de equipamentos elétricos (ar-condicionado, chuveiro, aquecedor, etc) de um ambiente com o intuito de reduzir o consumo de energia dos mesmos, desligando-os se necessário. O controle é feito através da analise do kWh consumido e registrado no relógio medidor de luz da unidade consumidora.

O controle será efetuado através de um microcontrolador que irá receber a informação via wifi e irá saber se a carga pode ser acionada ou não. Para determinar se a carga poderá ser acionada, será utilizado como parametro um limiar de kWh definido pelo usuário que irá ser comparado ao kWh atual da unidade consumidora, caso o consumo atual já seja maior ou igual ao limiar pré-definido, as cargas controladas pelo sistema não poderão ser acionadas pelo usuário, fazendo com que o usuário economize energia.

Para a captura do consumo atual, optamos pela opção menos invasiva a rede elétrica do usuário, não colocando nenhum tipo de equipamento na saída do relógio, e fazendo com que não seja necessário o acionamento de um serviço especializado para instalar o sistema. Para que isso seja possível será utilizada uma solução de visão computacional e uma camera IP. A câmera IP deve enviar uma imagem do relogio num intervalo pré definido de tempo para um servidor de aplicação. No servidor ao receber a imagem, será atrelado a unidade consumidora um valor numérico indicando o consumo em kWh , o valor será inferido da imagem a partir de um algoritmo de visão computacional. Ao obter o valor da imagem o sistema irá comparar o valor retirado da imagem ao limiar previamente configurado pelo cliente, caso maior ou igual o servidor acionará o serviço de atuação na rede. A ilustração desse cenário é mostrado na Figura 1.

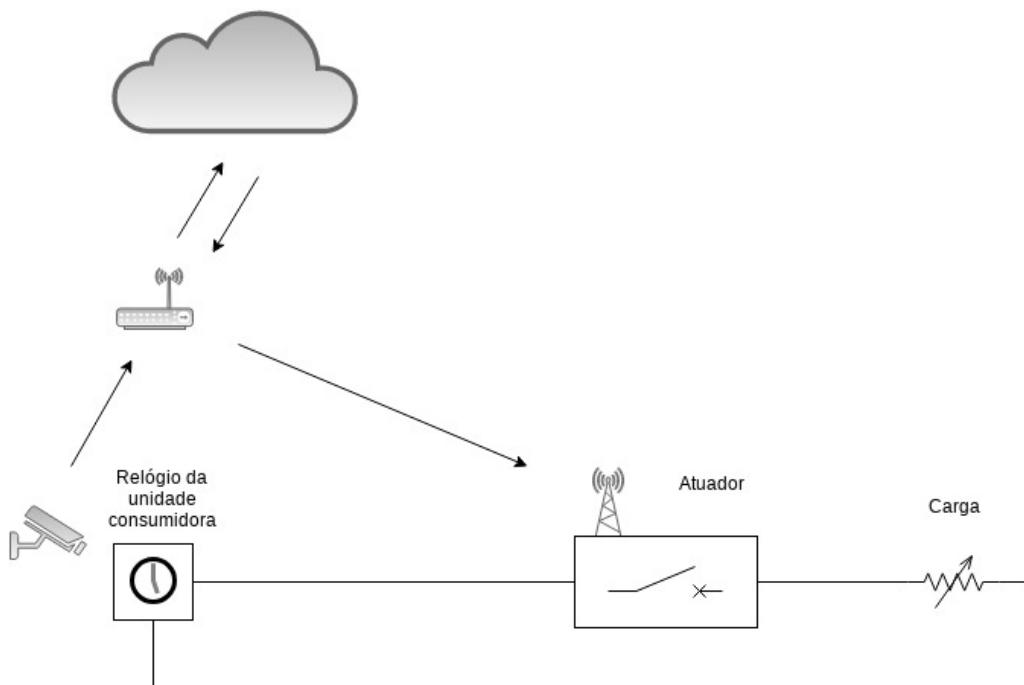


Figura 1: Diagrama da proposta.

2 Especificação

Para a realização da proposta descrita, será utilizado o microcontrolador de baixo custo **ESP8266 NodeMcu ESP12E** (ver Figura 2 e 3) que possui um módulo de alta performance para aplicações envolvendo wifi, fazendo com que não seja necessário nada além da própria placa de desenvolvimento para a parte de atuação do sistema, além disso o ESP8266 conta com um baixíssimo consumo de energia, fazendo com que não influencie na rede elétrica do usuário. Com 4 MB de memória flash, o ESP8266 permite criar variadas aplicações para projetos de IoT, acesso remoto, web servers e dataloggers, entre outros e possui suporte ao micropython o que permite utilizar várias das features da linguagem python o que deve aumentar a velocidade do desenvolvimento e tornar o suporte e evolução da solução mais simples.

Além do ESP8266, será utilizado uma camera IP, essa camera será a responsável por capturar a imagem do contador do relógio na unidade consumidora e mandará essa imagem para um servidor que irá reconhecer os dígitos através de um algoritmo de visão computacional.

A parte do servidor pode ser implementada utilizando serviços de computação em nuvem ou um servidor próprio. Para fins de protótipo será utilizado um servidor próprio para facilitar o desenvolvimento da solução. Mas quando finalizado, será implementado na nuvem.

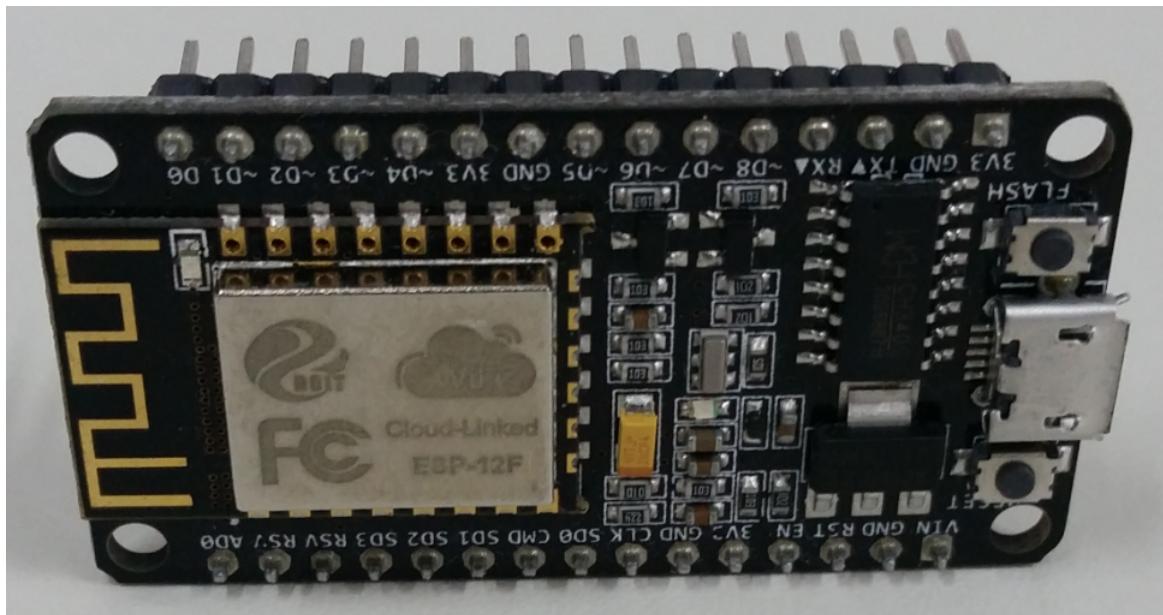


Figura 2: ESP8266 NodeMcu ESP12E visão frontal.



Figura 3: ESP8266 NodeMcu ESP12E visão traseira.

3 Desenvolvimento

O desenvolvimento do projeto ficou dividido em implementação do web-service, rotas e banco de dados, processamento OCR (Optical Character Recognition) da imagem no web-service, instalação do web-service na nuvem, cadastro da unidade consumidora e limite do kWh no web-service, captura de imagem por câmera IP e envio para o web-service e configuração e acesso do ESP8266 ao web-service. O responsável por cada parte ficou dividido da seguinte maneira:

- Implementação do web-service, rotas e banco de dados - (**Lucas**)
- Processamento OCR da imagem no web-service - (**Mário Allan**)
- Instalação do web-service na nuvem - (**Mário Allan**)
- Cadastro da unidade consumidora e limite do kWh no web-service - (**Mário Allan**)
- Captura de imagem por câmera IP e envio para o web-service - (**Mário Allan**)
- Configuração e acesso do ESP8266 ao web-service - (**Lucas/Mário Allan**)

4 Implementação do web-service, rotas e o banco de dados

O desenvolvimento da API foi feito em Flask e python e foram implementadas três rotas. Uma rota de cadastro de limites onde o usuário poderia cadastrar sua unidade consumidora e o limite em KWh que sua residencia deveria começar a diminuir o consumo de energia, essa rota possui as seguintes características:

```

1
2 Método: POST
3 URL: /limit
4 Body: { consumer_unity: String, limit: Integer }
```

A lógica implementada nessa rota é, caso a unidade consumidora exista, o limite deve ser substituído, caso contrário a unidade consumidora deve ser cadastrada junto ao seu limite inicial.

A segunda rota implementada é a rota para envio da imagem, é essa rota que a câmera deve usar quando adquirir uma imagem do usuário. Quando a imagem é adquirida pelo servidor ela é processada utilizando o OCR - como descrito na próxima seção - e identifica o numero descrito no relógio do usuário, esse número é comparado ao limite cadastrado para aquela unidade consumidora.

```

1
2 Método: POST
3 URL: /image_update
4 Body: { consumer_unity: String } and image
```

E a terceira rota implementada, é a rota responsável por verificar a ultima medida do relógio do usuário cadastrado na unidade consumidora enviada, a ultima medida identificada no relógio é comparada ao limite imposto pelo usuário para a sua unidade consumidora, caso a medição esteja abaixo do limite é enviado a string "*true*", indicando que a carga não precisa ser interrompida, caso a medição seja maior ou igual ao valor limite é enviado a string "*false*", indicando que a carga deve ser desligada [1], [13].

```

1
2 Método: GET
3 URL: /check
4 Body: { consumer_unity: String }
```

Com a implementação dessas três rotas é possível controlar o consumo de cada unidade consumidora através do controle das cargas críticas para o usuário.

No banco de dados foi implementado duas tabelas, a tabela **Costumer** que recebe as informações de cadastro, isto é, o nome da unidade consumidora e o valor limite de kwh. E a tabela **Measurement** que recebe as informações do processamento OCR da imagem da unidade consumidora escolhida, o caminho de armazenamento da imagem recebida pelo web-service e o id da unidade consumidora escolhida que foi cadastrado na tabela **Costumer**, pois esse item tem relação entre as duas tabelas.

5 Processamento OCR da imagem no web-service

Para executar o **processamento de OCR** (Optical Character Recognition) é utilizado a ferramenta **Tesseract** que é um mecanismo de reconhecimento de texto (OCR), que suporta uma ampla variedade de idiomas, no qual extrai textos impressos em imagens. Além dessa ferramenta, também é usado as bibliotecas python **Pytesseract**, **Numpy** e **Pillow**. Com esses componentes em funcionamento, foi implementado o código abaixo na rota do web-service que recebe a imagem (`/image_update`). No código é feito a abertura da imagem na variável **img** que utiliza a bilbioteca **Pillow**. Em seguida a variável **img** é passada para a variável **value** que contém o método **ocr** da biblioteca **Pytesseract** e que chama a função **image_to_string**, no qual recebe como parâmetro a imagem a ser processada, o idioma que vai ser comparado os textos extraídos da imagem para conversão em string, no qual foi configurado a **linguagem em inglês (lang='eng')** pois como vai ser convertido números, tanto em inglês como em português os números são os mesmos, então não faz diferença nesse parâmetro, e a configuração **psm (Page segmentation modes) modo 10 (config='--psm 10')** que trata a imagem como um único caractere, o que garante que a conversão de uma imagem com números para string seja mais precisa [2], [5], [6].

```

1 import pytesseract as ocr
2 from PIL import Image
3
4 img =Image.open(image) // ABRE A IMG REFERIDA
5
6 //Function to get the measurement value from the image
7 value = ocr.image_to_string(img, lang='eng', config='
8 --psm 10') // CONVERTE A IMG PARA STRING COM OS
               PARAMETROS DE LINGUA INGLESA E PSM 10
print(value)

```

6 Instalação do web-service na nuvem

Para o funcionamento do web-service na nuvem do google (Figura 4) no endereço de acesso **http://35.211.13.184**, foi configurado o servidor web (HTTP e IMAP/-POP3/Proxy) **Nginx** (Figura 5), no qual dentre as principais configurações, foi o funcionamento por **HTTP (porta 80)** e a liberação da **porta de execução do web-service (porta 5000)** [9].



Figura 4: Nuvem do google.



Figura 5: Servidor web (HTTP e IMAP/POP3/Proxy) Nginx.

Após as configurações, o web-service é implementado na nuvem e em seguida é executado, iniciando o envio e recebimento das informações. O web-service está configurado com o **modo DEBUG** ativado, no qual permite visualizar as mensagens de **POST** e **GET** recebidas por ele. E para verificar as informações adicionadas no banco de dados, é utilizado o software de linha de comando **sqlite** [3]. Essa informações são ilustradas na Figura 6.

The screenshot shows two terminal windows. The top window, titled 'Terminal', displays log entries from a web-service. The bottom window, also titled 'Terminal', shows a SQLite database query and its results.

```

Atividades Terminal
root@vm-mario2: /home/mario
Arquivo Editar Ver Pesquisar Terminal Ajuda
nit=Barra HTTP/1.0" 200 -
127.0.0.1 - - [09/Jul/2019 20:39:33] "GET /check?consumer_u
nit=Barra HTTP/1.0" 200 -
127.0.0.1 - - [09/Jul/2019 20:39:37] "POST /limit HTTP/1.0"
200 -
127.0.0.1 - - [09/Jul/2019 20:39:44] "GET /check?consumer_u
nit=Barra HTTP/1.0" 200 -
[]

root@vm-mario2: /tmp
Arquivo Editar Ver Pesquisar Terminal Ajuda
SELECT * FROM costumer
...> ;
id      consumer_unit  consumption_limit
-----  -----
1       Barra          14786
2       Barra          Barra
sqlite> []

```

Figura 6: Exibição do modo DEBUG do web-service (1) e das informações do banco de dados mostradas pelo sqlite (2).

7 Cadastro da unidade consumidora e limite do kWh no web-service

Para enviar o nome da unidade consumidora e o valor limite de kWh para o web-service que está rodando na nuvem no endereço <http://35.211.13.184>, é utilizado um **arquivo em python** mostrado abaixo com a biblioteca **Requests** que é uma biblioteca HTTP do Python. Com o Request, é utilizado o metodo HTTP de envio **POST**. No metodo é configurado o endereço de envio, composto pelo endereço e a rota destinada ao recebimento do nome da unidade consumidora e o valor limite de kWh (`url = http://35.211.13.184/limit`), e a variável (`json`) que armazena as variáveis (`unit`) e (`limit`) que carregam respectivamente o nome da unidade consumidora e o valor limite de kWh [11], [7], [4].

```

1 import requests
2
3 #coding: utf-8
4
5 print("====")
6 print("----- INTERFACE DE CADASTRO -----")
7 print("====")
8 print("")
9 unit = input("Informe a Unidade Consumidora: ")
10 limit = input("Informe o Valor Limite do kW/h: ")
11 print("")
12 print("-----")
13 print("Unidade e limite cadastrado com sucesso")
14 print("-----")
15 print("")
16
17 url='http://35.211.13.184/limit'

```

7 CADASTRO DA UNIDADE CONSUMIDORA E LIMITE DO KWH NO WEB-SERVICE

```
18 json={"consumer_unity":unit, "limit":limit}  
19  
20 requests.post(url,json=json)
```

A ilustração desse cenário é exibido nas Figuras 7 e 8.

mario@mario: ~/allan_eng/pji3/nuvem_google/limit_e_img\$ Arquivo Editar Ver Pesquisar Terminal Ajuda ===== Informe a Unidade Consumidora: Barra Informe o Valor Limite do kW/h: 14786 ----- Unidade e limite cadastrado com sucesso ----- mario@mario:~/allan_eng/pji3/nuvem_google/limit_e_img\$ mario@mario:~/allan_eng/pji3/nuvem_google/limit_e_img\$

Figura 7: Execução do arquivo python que cadastra a unidade consumidora e o limite de kWh e envia pro web-service.

Atividades Terminal root@vm-mario2: /home/mario Arquivo Editar Ver Pesquisar Terminal Ajuda nit=Barra HTTP/1.0" 200 - 127.0.0.1 - - [09/Jul/2019 20:39:33] "GET /check?consumer_u nit=Barra HTTP/1.0" 200 - 127.0.0.1 - - [09/Jul/2019 20:39:37] "POST /limit HTTP/1.0" 200 - 127.0.0.1 - - [09/Jul/2019 20:39:44] "GET /check?consumer_u nit=Barra HTTP/1.0" 200 -

root@vm-mario2: /tmp Arquivo Editar Ver Pesquisar Terminal Ajuda SELECT * FROM costumer ...> ; id consumer_unit consumption_limit ----- 1 Barra 14786 2 Barra

Figura 8: Log do web-service (1) e do bando de dados com o sqlite (2) pelo recebimento do cadastro da unidade consumidora e o limite de kWh.

8 Captura de imagem por câmera IP e envio para o web-service

8.1 Captura de imagem por câmera IP

Os componentes utilizados para a captura da imagem através da câmera IP e envio para o web-service são compostos por uma Webcam ligada há um computador com interface ethernet e wifi rodando o sistema operacional Linux Ubuntu 18.04.

Com o cenário em funcionamento, a webcam captura a imagem do relógio da unidade consumidora. Na captura da imagem do relógio, a informação de interesse é o valor do kWh consumido até o momento. Dessa forma, é recortado essa informação e gerado uma nova imagem.

Para a captura da imagem pela webcam e recorte da informação de interesse, são utilizados os respectivos software de linha de comando do linux:

- Fswebcam
- Imagemagick (Ferramentas convert e crop)

Na captura da imagem pelo software **fswebcam** é configurado uma resolução de 640x480 que foi considerado uma boa resolução para a obtenção de uma imagem de qualidade além de gerar um arquivo de tamanho pequeno [12].

```
1 fswebcam -d /dev/video0 -r 640x480 --jpeg 85 -D 1 /tmp/foto.jpg
```

Após capturar a imagem, com as ferramentas **convert** e **crop** do software **ImageMagick**, é recortado apenas a informação de interesse, que é o valor do kWh consumido no atual momento pela unidade consumidora. Para esse recorte é configurado uma janela de corte de **140x30** com movimentação horizontal e vertical da imagem respectivamente de **+250+300** [8].

```
1 convert /tmp/foto.jpg -crop 140x30+250+300 /tmp/mg.jpg
```

8.2 Envio da imagem para o web-service

Para enviar a imagem para o web-service que está rodando na nuvem no endereço **http://35.211.13.184**, é utilizado um **arquivo em python** mostrado abaixo com a biblioteca **Requests** que é uma biblioteca HTTP do Python. Com o Request, é utilizado o metodo HTTP de envio **POST**. No metodo é configurado o endereço de envio, composto pelo endereço, a rota destinada ao recebimento de imagem no web-service e o nome do responsável da unidade consumidora monitorada (**url = http://35.211.13.184/image_update?consumer_unit=Barra**), e a variável (**files**) que carrega a abertura de leitura da imagem escolhida que se encontra em um determinado caminho (**/tmp/img.jpg**) [11], [7], [4].

```
1 import requests
2
3 end='http://35.211.13.184/image_update',
4 files={'image': open('/tmp/img.jpg', 'rb')}
```

```

4 unit = '?consumer_unit=Barra'
5 url = end + unit
6 img = '/tmp/img.jpg'
7 files={'file': open(img,'rb')}
8
9 requests.post(url,files=files)

```

Com esses três itens finalizados, é implementados um **shell script** para otimizar execução desses processos.

```

1 #!/bin/bash
2
3 fswebcam -d /dev/video0 -r 640x480 --jpeg 85 -D 1 /tmp/foto.
4   jpg
5 convert /tmp/foto.jpg -crop 140x30+250+300 /tmp/img.jpg
6
7 python3 /tmp/envia_img_e_unidade_consumidora_WS.py

```

A ilustração de teste desse cenário é exibido nas Figuras 9, 10 e 11.



Figura 9: Simulador genérico em papel de relógio de luz com o valor kWh de consumo de "14786".

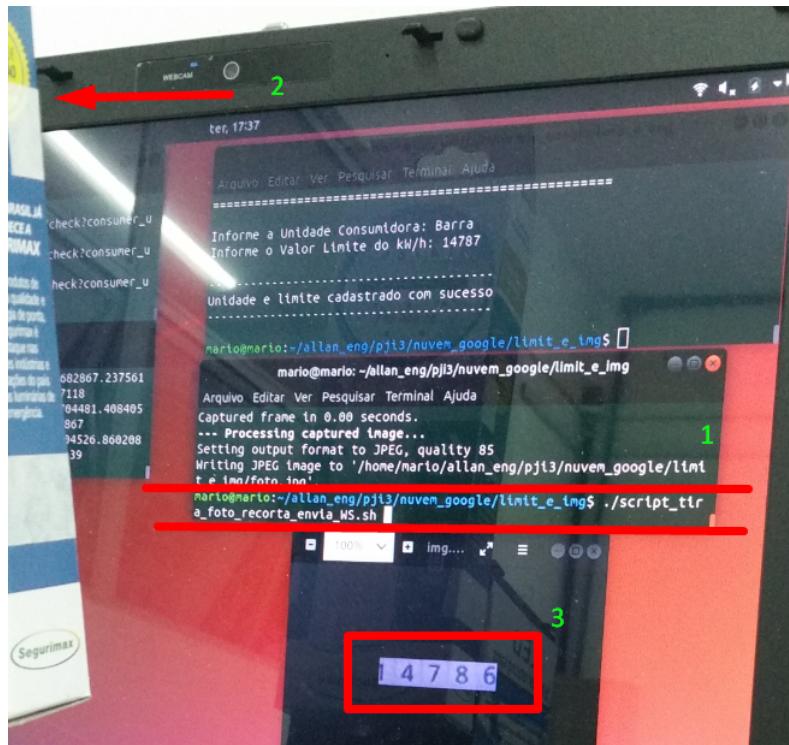


Figura 10: Execução do script (1) que Captura (2), recorta (3) e envia a imagem do simulador de papel do relógio de luz para o web-service.

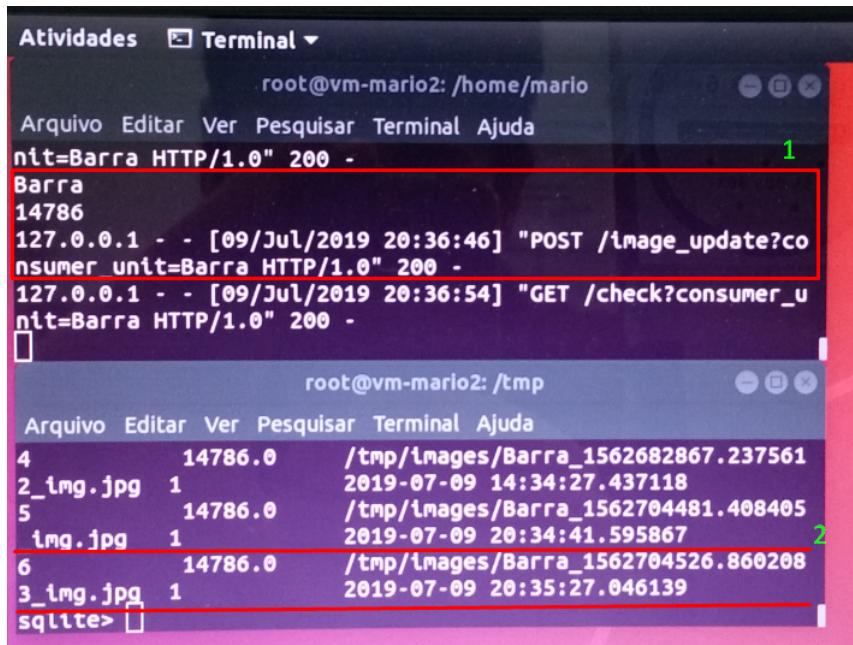


Figura 11: Log do web-service (1) e do bando de dados (2) pelo recebimento da imagem enviada.

9 Configuração e acesso do ESP8266 ao web-service

A implementação do código da ESP8266 que acessa o web-service para verificar o valor de leitura do kWh com o valor cadastrado, inicialmente seria feito em micro python, mas devido a falta de tempo foi implementando com a linguagem de programação padrão do Arduino, a linguagem C++. No código abaixo foi implementado um **cliente HTTP** que em resumo após a conexão com uma rede wifi, usa o método **GET** para acessar o endereço e a rota (<http://35.211.13.184/check>) do web-service que esta rodando na nuvem. Nesse acesso também é informado o nome do responsável da unidade consumidora, que caso esse nome exista no banco de dados do web-service, retornar uma **mensagem True ou False**. Se o valor de leitura do kWh for menor que o valor cadastrado, então é retornado a mensagem **True**, **senão** retorna a de **False**. Com o retorno dessas mensagens é configurado o acendimento ou apagamento do Led da ESP8266 na **Porta 2** (Figura 12). Assim **caso** a mensagem seja **True**, então acende o Led da ESP8266 (Figura 13). **Caso** a mensagem seja **False** então apaga o Led da ESP8266 (Figura 14). **Caso** a mensagem seja qualquer outro valor, então também acende o Led da ESP8266 (Figura 13).

No código foi configurado no teste de **caso** da mensagem apenas o primeiro bit (vetor 0) pois ja é o suficiente para a identificação a diferença entre as mensagens, e também uma repetição (Loop) de acesso da ESP8266 ao web-service, no qual a cada 10 segundos é verificado se houve alteração no valor da leitura do kWh com o valor cadastrado para garantir que o corte ou a continuação do fornecimento da energia não demore muito para ser testado [10].

```

1 #include <Arduino.h>
2 #include <ESP8266WiFi.h>
3 #include <ESP8266WiFiMulti.h>
4 #include <ESP8266HTTPClient.h>
5 #include <WiFiClient.h>
6
7 ESP8266WiFiMulti WiFiMulti;
8
9 const char* ssid = "Nome_da_rede_wifi";
10 const char* password = "Senha_da_rede_wifi";
11
12 void setup() {
13
14     Serial.begin(115200);
15     Serial.println();
16     Serial.println();
17     Serial.println();
18
19     WiFi.mode(WIFI_STA);
20     WiFi.begin(ssid, password);
21     Serial.println("");
22
23 // Wait for connection
24     while (WiFi.status() != WL_CONNECTED) {
```

```
25     delay(500);
26     Serial.print(".");
27 }
28 Serial.println("");
29 Serial.print("Connected to ");
30 Serial.println(ssid);
31 Serial.print("IP address: ");
32 Serial.println(WiFi.localIP());
33
34 pinMode (2, OUTPUT);
35 }
36
37 void loop() {
38
39     String payload;
40
41 // wait for WiFi connection
42 if ((WiFiMulti.run() == WL_CONNECTED)) {
43
44     WiFiClient client;
45     HTTPClient http;
46
47     Serial.print("[HTTP] begin...\n");
48     if (http.begin(client, "http://35.211.13.184/check?
49         consumer_unit=Barra")) {
50
51         Serial.print("[HTTP] GET...\n");
52         // start connection and send HTTP header
53         int httpCode = http.GET();
54
55         // httpCode will be negative on error
56         if (httpCode > 0) {
57             // HTTP header has been send and Server response
58             // header has been handled
59             Serial.printf("[HTTP] GET... code: %d\n", httpCode);
60
61             // file found at server
62             if (httpCode == HTTP_CODE_OK || httpCode ==
63                 HTTP_CODE_MOVED_PERMANENTLY) {
64                 payload = http.getString();
65                 Serial.println(payload);
66             }
67         } else {
68             Serial.printf("[HTTP] GET... failed, error: %s\n",
69             http.errorToString(httpCode).c_str());
70         }
71
72         http.end();
73 }
```

```
70     } else {
71         Serial.printf("[HTTP] Unable to connect\n");
72     }
73 }
74
75 switch (payload[0])
76 {
77     case 't':
78         digitalWrite(2, LOW);
79         break;
80
81     case 'f':
82         digitalWrite(2, HIGH);
83         break;
84
85     default:
86         digitalWrite(2, LOW);
87 }
88
89 delay(10000);
90 }
```

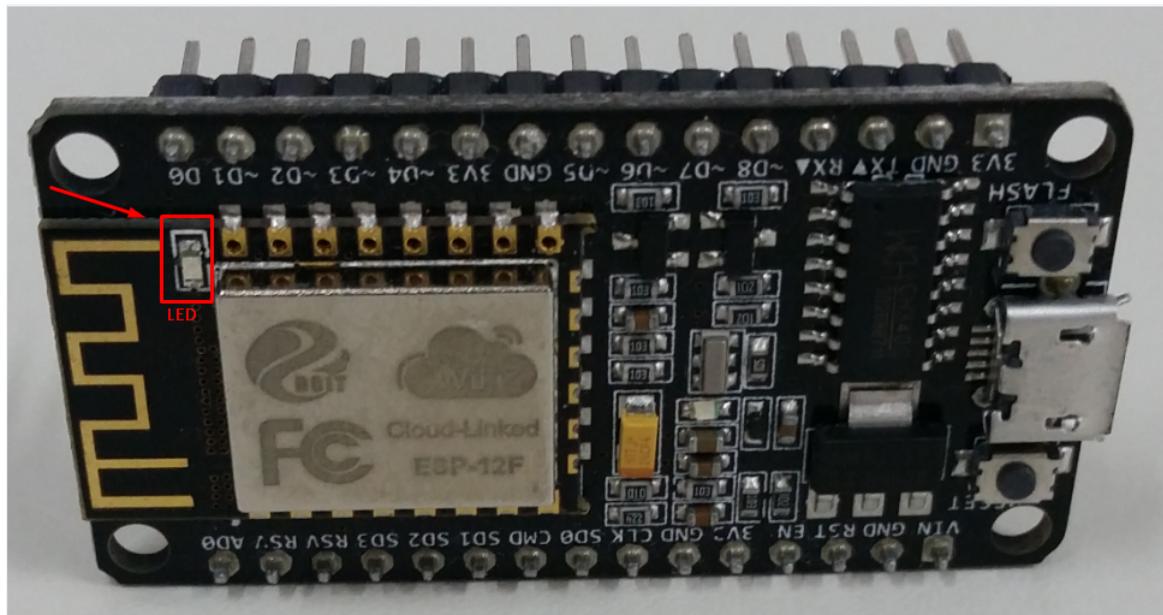


Figura 12: LED do ESP8266 (Porta 2) utilizado na simulação do fornecimento ou corte da energia.

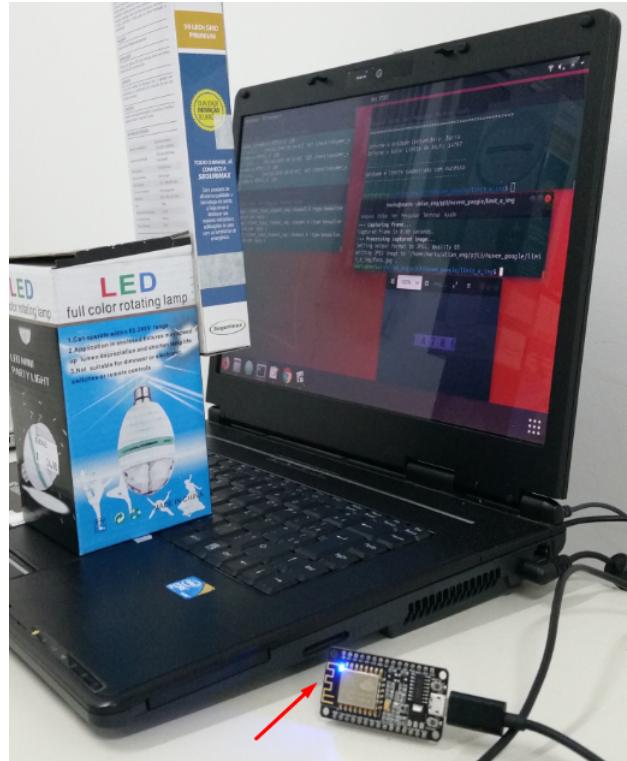


Figura 13: ESP8266 com o LED aceso simulando a liberação de energia da unidade consumidora após verificar que o valor de leitura do kWh está menor que o valor cadastrado (Mensagem True).



Figura 14: ESP8266 com o LED apagado simulando o corte de energia da unidade consumidora após verificar que o valor de leitura do kWh está maior ou igual ao valor cadastrado (Mensagem False).

10 Conclusão

Através dessa implementação de IoT é possível controlar facilmente cargas críticas em uma unidade consumidora, de uma forma dinâmica e personalizada, reduzindo o consumo e os gastos com energia.

O uso da API é simples, sendo a única ação do usuário cadastrar o seu limite e sua unidade consumidora. A ideia é que isso poderia ser aplicado de forma pública para que os clientes das empresas de fornecimento de energia elétrica pudessem controlar o uso de sua energia, suspendendo cargas críticas conforme a necessidade de forma automática.

Esse sistema também se mostra útil para donos de redes de lojas ou de uma série de estabelecimentos, pois os mesmos poderiam controlar o consumo de seus estabelecimentos automaticamente, tornando mais simples a redução de custos de energia.

Uma sugestão para temas futuros seria aprimorar o sistema para que ele faça recomendações de desligamento de acordo com a taxa dinâmica da rede, e que monitore o uso de cargas de interesse em horários onde as cargas deveriam estar desligadas.

Referências

- [1] <http://flask.pocoo.org>.
- [2] <https://blog.codeexpertslearning.com.br/lendo-imagens-uma-abordagem-%C3%A0-ocr-com-tesseract-e-python> ee8e8009f2ab.
- [3] <https://cadernodelaboratorio.com.br/2016/01/11/sqlite3-ferramenta-de-linha-de-comando-para-acesso-a-bancos-sqlite/>.
- [4] [https://en.wikipedia.org/wiki/Requests\(*software*\)](https://en.wikipedia.org/wiki/Requests_(software)).
- [5] <https://github.com/tesseract-ocr/tesseract/wiki>.
- [6] [https://stackoverflow.com/questions/44619077/pytesseract-ocr-multiple-config options](https://stackoverflow.com/questions/44619077/pytesseract-ocr-multiple-config-options).
- [7] [https://stackoverflow.com/questions/8634473/sending-json-request-with python](https://stackoverflow.com/questions/8634473/sending-json-request-with-python).
- [8] <https://tadeubento.com/2016/bash-imagemagick-cortar-imagens/>.
- [9] <https://www.digitalocean.com/community/tutorials/how-to-configure-nginx-as-a-web-server-and-reverse-proxy-for-apache-on-one-ubuntu-18-04-server>.
- [10] <https://www.filipeflop.com/blog/programar-nodemcu-com-ide-arduino/>.
- [11] [https://www.geeksforgeeks.org/get-post-requests-using python/](https://www.geeksforgeeks.org/get-post-requests-using-python/).
- [12] [https://www.vivaolinux.com.br/dica/fswebcam-Tirando-foto-pelo terminal](https://www.vivaolinux.com.br/dica/fswebcam-Tirando-foto-pelo-terminal).
- [13] <https://www.youtube.com/channel/UCiHEeTXhVQDnw4m8OVI36yA>.