



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
GRADUAÇÃO EM ENGENHARIA MECATRÔNICA
SISTEMAS EMBARCADOS II
PROF. DR. ÉDER ALVES DE MOURA



TRABALHO 2
RELATÓRIO

EDUARDO CUNHA DE CARVALHO - 11621EMT003
JOÃO ALBERTO ANDRAUS GASSAN - 11211EMT003
JOÃO VICTOR MEDEIROS ROCHA - 41621ETE005
MATHEUS ALVES DE PAULA - 11521EMT008
NICOLAS FISCHMANN - 12011EMT032

UBERLÂNDIA
MARÇO DE 2022

SUMÁRIO

1. OBJETIVO	2
2. INTRODUÇÃO	3
3. DESENVOLVIMENTO	3
4. RESULTADOS	8
5. CONCLUSÃO	9
6. ANEXO COM O CÓDIGO	10
7. BIBLIOGRAFIA	16

1. OBJETIVO

O objetivo deste trabalho é o desenvolvimento de uma aplicação considerando o cenário de Internet das Coisas (Internet of Things – IoT), que é um mercado de trabalho em crescente demanda.

Assim, desenvolve-se uma aplicação com Linux + Python + Flask + Arduino/Simulação com o intuito de representar o controle de uma casa automatizada, onde:

- A aplicação é executada no ambiente virtual do Linux;
- Desenvolve-se a interface Homem-Máquina em Python + Flask (web server), que permitirá controlar a função Liga/Desliga de equipamentos da casa. Essa interface será acessada em um navegador executando na máquina hospedeira e não na própria máquina virtual;
- Para representar os equipamentos, desenvolve-se um aplicativo em que os comandos de controle serão enviados via conexão socket.

2. INTRODUÇÃO

A Internet das Coisas, em inglês, Internet of Things (IoT) é uma referência à habilidade de diferentes tipos de objetos conseguirem estabelecer conexão com a internet, desde eletrodomésticos até carros. Portanto, esses itens conseguem coletar e transmitir dados a partir da nuvem.

Atualmente, já é possível encontrar dispositivos IoT sendo utilizados tanto em situações comuns da vida diária como no âmbito profissional. Desse modo, essas ferramentas tecnológicas estão contribuindo para o acontecimento da transformação digital no mundo.

Além disso, programas em linguagem Python têm conquistado o mercado e alcançou programadores fiéis ao redor do mundo, não podendo negar que a linguagem alcançou um patamar elevado e muitos projetos e empresas têm apostado na linguagem em seus novos projetos.

É uma linguagem que prioriza a legibilidade do código, combinando uma sintaxe concisa e clara. Com seus recursos poderosos de sua biblioteca padrão e por muitos módulos e frameworks.

3. DESENVOLVIMENTO

Primeiramente foi criado um esquema de como o projeto foi montado e como funciona o fluxo da aplicação para que fosse possível entendermos melhor como dará o funcionamento da aplicação.

Uma vez que entendido e bem definido é possível criar uma aplicação sólida, organizada e que tenha uma arquitetura limpa semelhante ao clean code, além de construir todo o projeto utilizando boas práticas.

Sendo assim veja abaixo como ficou o escopo o diagrama lógico e o fluxo da aplicação:

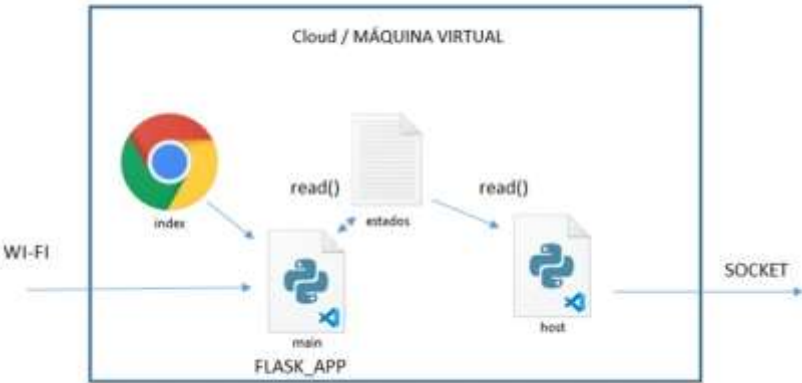


Figura 1 – Escopo e fluxo da aplicação.

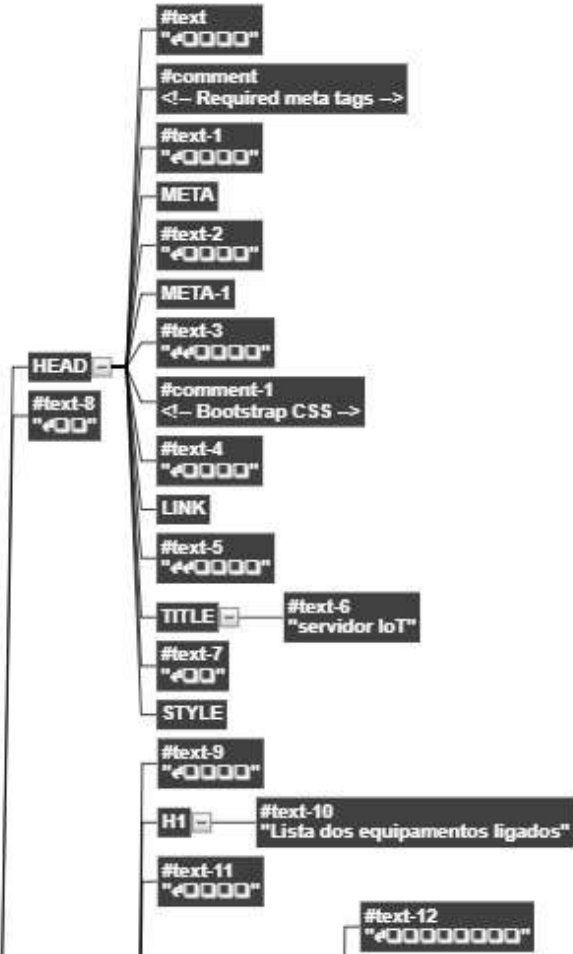


Figura 2 – Diagrama lógico da aplicação parte I.

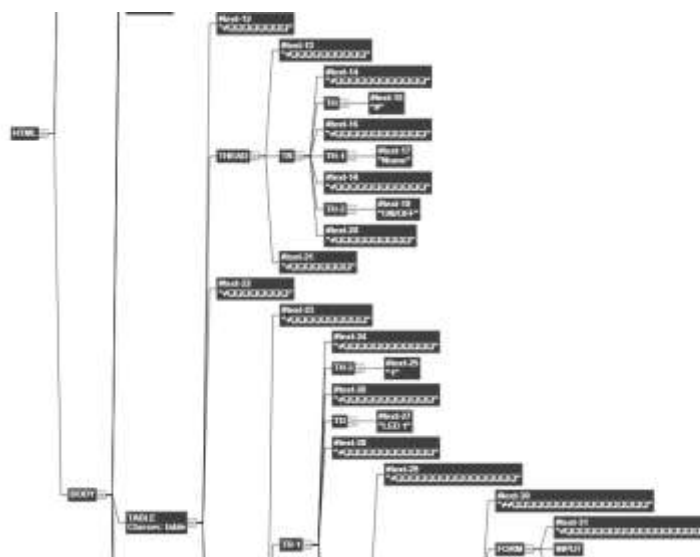


Figura 3 – Diagrama lógico da aplicação parte II.

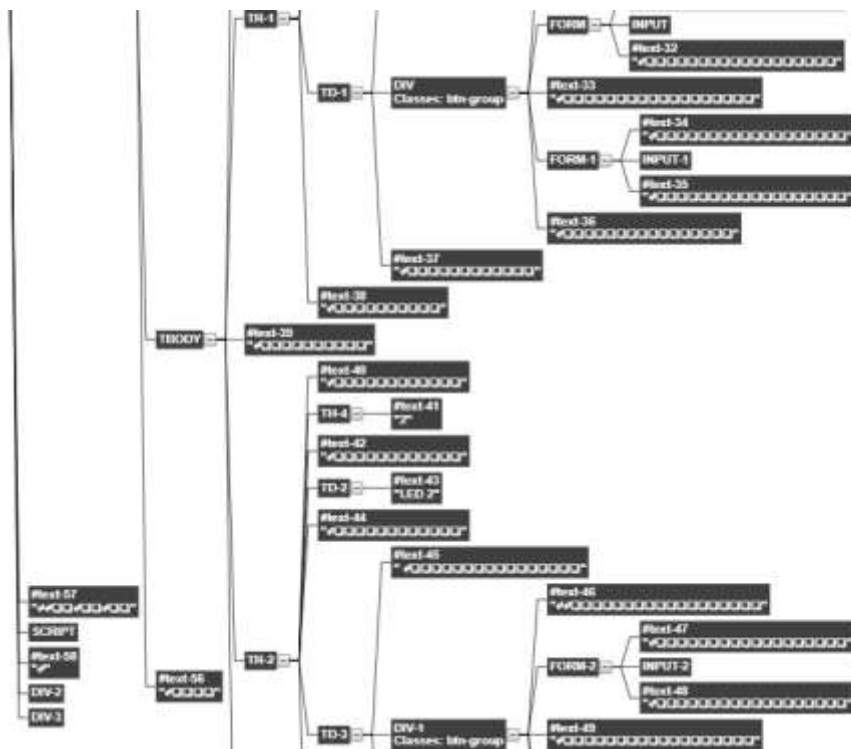


Figura 4 – Diagrama lógico da aplicação parte III.

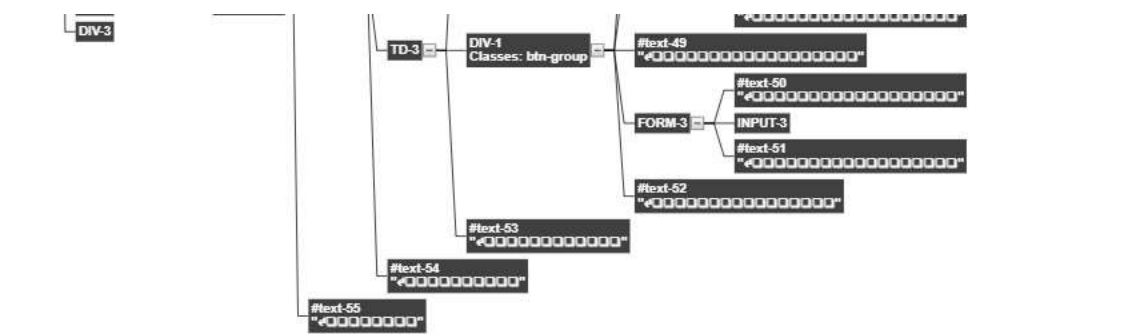


Figura 5 – Diagrama lógico da aplicação parte IV.

Visto o esquema da figura 1 que pode ser vista acima, a nossa aplicação pode ser dividida em 3 partes ou etapas. Sendo elas:

1. Desenvolvimento da Interface (Front-End - Index);
2. Desenvolvimento das APIs, para a VM (Back-End - FLASK_APP);
3. Simulação de Leds que acendem através do Socket (Cliente - Socket);

Agora com o escopo do projeto bem detalhado, foi possível trabalhar e construir cada parte do projeto, portanto vamos apresentar como se deu a construção das etapas, Front-End, Back-End e Cliente.

DESENVOLVIMENTO DA INTERFACE (FRONT-END)

O desenvolvimento da interface se deu com a “linguagem” de marcação HTML em conjunto com a “linguagem” de estilização CSS. As linguagens, HTML e CSS são muito utilizadas quando o assunto é desenvolvimento web. Elas são capazes, juntas de criarem o que é nomeado no mundo do desenvolvimento como FRONT-END ou simples de View que é o local onde o cliente consegue de fato interagir com a aplicação pois elas juntas conseguem renderizar na tela diversas informações.

Sendo assim, foi possível criar toda a página e tornar ela esteticamente mais bonita para que o usuário consiga visualizar uma interface ao mesmo tempo simples, funcional e intuitiva, além de seguir alguns padrões de UX/UI.

Para dar suporte ao CSS utilizamos uma biblioteca que auxilia na estilização que é a biblioteca do Bootstrap. Com isso foi possível construir o Front - End contendo 2 pares de botões ON/OFF para controlar o estado dos LEDs ou “lâmpada” que o usuário quer ligar ou desligar veja abaixo como ficou o Front-END da aplicação:

Lista dos equipamentos ligados		
#	Nome	ON/OFF
1	LED 1	ON OFF
2	LED 2	ON OFF

Figura 6 – Front-END da aplicação.

Portanto com a interface pronta e o par de botões ON/OFF funcionando temos toda a primeira etapa do nosso escopo completo e agora devemos ir em busca da segunda etapa e construir o nosso Back-End que é quem vai receber as informações do estado de click deste botão e enviar para o nosso Socket.

DESENVOLVIMENTO DOS END-POINTS e APIs (BACK-END)

Para que a aplicação seja funcional é necessário ter um BACK-END um servidor ou até mesmo uma máquina virtual para onde vamos enviar informações através de requisições.

Para o nosso projeto isso não foi diferente. Portanto, para construir nossos End-points utilizamos a linguagem de programação Python com a biblioteca Flask que é uma biblioteca que auxilia na criação de APIs REST que depois podem ser consumidas.

Para a aplicação ser funcional utilizamos uma lógica que consiste no seguinte:

Para ser possível que o usuário mude o estado do LED para ON ou OFF é necessário disparar um evento `onClick()` como sendo um parâmetro da nossa tag `<button>` que é a responsável por desenhar no FRONT-END o nosso botão. Quando o evento ocorrer irá invocar ou realizar a chamada de uma função que executa uma ação.

Entretanto, além de disparar o evento `onClick()` é necessário enviar o estado do botão para os nossos end points pois são eles que posteriormente, irão informar para o Socket qual ação deve ser executada ligar o LED ou desligar o LED.

Entendido isso vamos discorrer sobre como funciona para a interface enviar e se comunicar com o END Point criado em Flask. Para realizar a comunicação entre a nossa interface do usuário com o End point utilizamos a Arquitetura de Microserviço CRUD que na verdade pode ser vista como arquitetura ou apenas como um acrônimo.

O CRUD ou CREATE, READ, UPDATE e DELETE segue um padrão de requisições HTTP para que seja possível comunicar o front com o back-end.

Para que isso fosse possível dentro da nossa aplicação foram construídas ROTAS onde eles eram responsáveis por receber essas ações e realizar o fluxo da aplicação. Por conveniência não criamos nenhuma URL especial para o usuário pois isto não foi um parâmetro necessário e nem definido dentro do escopo do projeto.

Logo para que aquele botão se comunica-se com o back e posteriormente com o socket “embedamos” eles através de uma tag <form> ou formulário que recebe o parâmetro method = “POST, ou seja, o formulário irá enviar uma requisição do tipo POST para o end point feito isso a ultima etapa a ser construída é a etapa de consumir esses end points e ascender os LEDs através do Socket.


SIMULAÇÃO DE LEDs QUE LIGAM ATRAVÉS DO SOCKET

Portanto chega-se a ultima etapa do escopo definido para o projeto onde o nosso Back e Front- comunicados e construídos e rodando em uma máquina virtual através da porta 5000 consegue se comunicar com o Socket para os LEDs ligarem ou desligarem.

Para realizar essa ação de ligar ou desligar utilizamos a biblioteca Pygame para que essa animação de LED ligado ou desligado se ocorre. Entretanto para que quando o botão for pressionado pelo usuário na interface e para que o LED ligue ou desligue foi necessário por conveniência criar uma lista inicializando os parâmetros dos botões como sendo [0,0] ou seja estado de desligado, a partir do momento que o usuário pressione o botão na interface o botão irá alterar esta lista para [1,1] ou seja botão ligado e portanto foi possível construir a aplicação que se mostra funcional e muito intuitiva que posteriormente pode passar por mudanças e algumas melhorias. E os resultados podem ser conferidos abaixo na sessão resultados.

4. RESULTADOS

Portanto feito tudo isso, foi possível construir a aplicação funcional que ficou da seguinte maneira:



#	Nome	ON/OFF
1	LED 1	<input type="button" value="ON"/> <input type="button" value="OFF"/>
2	LED 2	<input type="button" value="ON"/> <input type="button" value="OFF"/>

Figura 7 – Interface da aplicação.

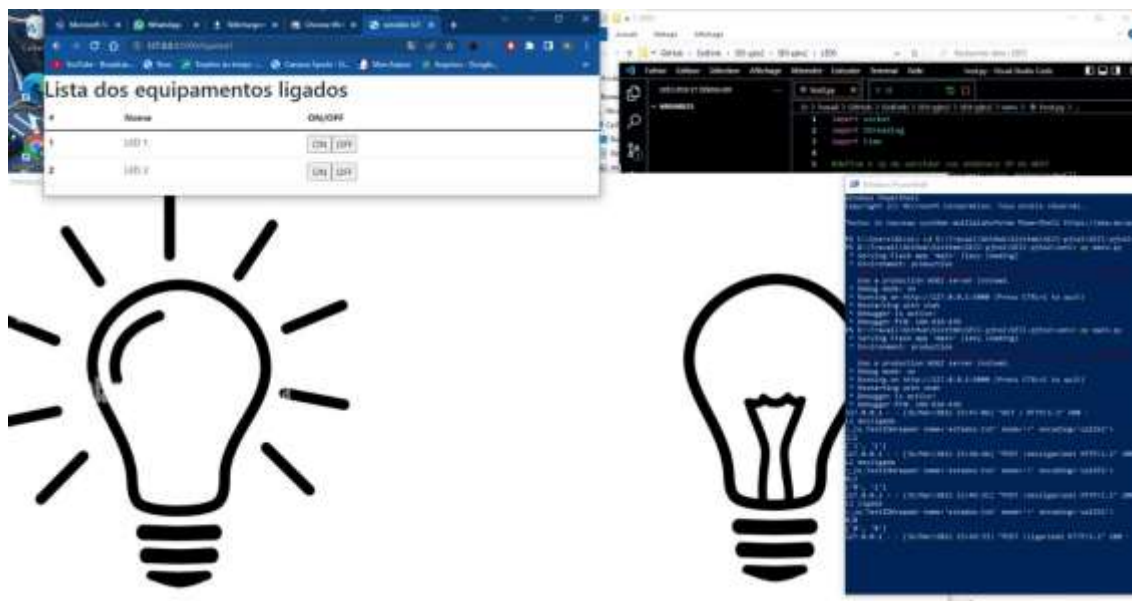


Figura 8 – Par LED 1 ligado e LED 2 desligado.

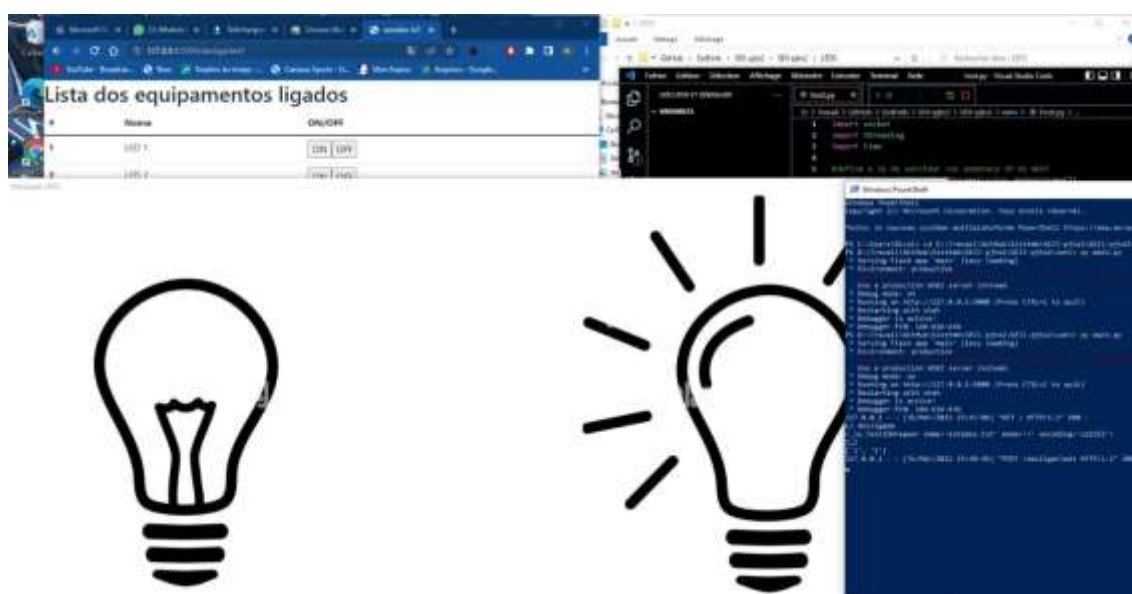


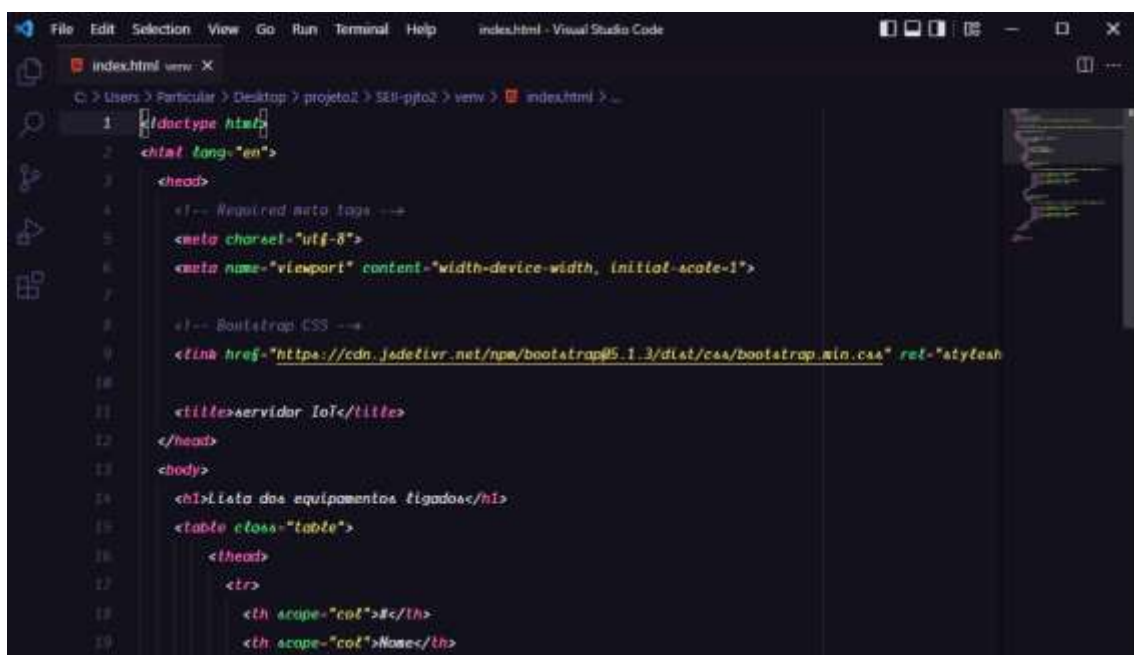
Figura 9 – Par LED 1 desligado e LED 2 ligado.

5. CONCLUSÃO

Com o presente trabalho foi possível observar que é possível desenvolver aplicações simples, robustas e funcionais e que são facilmente implementadas, gerando a posteriori um produto de valor que pode impactar trazendo conforto para vida de

diversas pessoas. Além disso, este projeto foi de grande aprendizado para todos os membros do grupo pois assim foi possível colocar em prática os conceitos aprendidos ao longo da disciplina de Sistemas Embarcados II. Entretanto como esta aplicação é uma aplicação que impacta positivamente algumas pessoas este produto em breve terá modificações para que se torne cada vez melhor.

6. ANEXO COM O CÓDIGO



```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <!-- Required meta tags -->
5   <meta charset="utf-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1">
7
8   <!-- Bootstrap CSS -->
9   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
10
11   <title>servidor IoT</title>
12 </head>
13 <body>
14   <h1>lista dos equipamentos ligados</h1>
15   <table class="table">
16     <thead>
17       <tr>
18         <th scope="col">#</th>
19         <th scope="col">Nome</th>
```

Anexo 1: index.html

```

19     <th scope="col">Nome</th>
20     <th scope="col">ON/OFF</th>
21 </tr>
22 </thead>
23 <tbody>
24 <tr>
25     <th scope="row">1</th>
26     <td>LED 1</td>
27     <td>
28         <div class="btn-group" role="group" aria-label="botao ligacao led 1">
29
30             <form action="/ligarled1" method="POST">
31                 <input type="submit" value = "ON">
32             </form>
33             <form action="/desligarled1" method="POST">
34                 <input type="submit" value = "OFF">
35             </form>
36         </div>
37     </td>
38 </tr>
39 <tr>
40     <th scope="row">2</th>
41     <td>LED 2</td>
42     <td>
43         <div class="btn-group" role="group" aria-label="botao ligacao led 2">

```

Anexo 2: index.html

```

43         <div class="btn-group" role="group" aria-label="botao ligacao led 2">
44
45             <form action="/ligarled2" method="POST">
46                 <input type="submit" value = "ON">
47             </form>
48             <form action="/desligarled2" method="POST">
49                 <input type="submit" value = "OFF">
50             </form>
51         </div>
52     </td>
53 </tr>
54 </tbody>
55 </table>
56
57 </body>
58 <script></script>
59 </html>
60

```

Anexo 3: index.html

```

1  from flask import Flask, render_template, request, redirect, url_for
2
3
4  app = Flask(__name__)
5
6  @app.route("/", methods = ["GET"])
7  def index():
8      return render_template("index.html")
9
10
11 @app.route("/ligarled1", methods = ["POST"])
12 def ligarled1():
13     print("L1 ligada")
14     modificar_estado(1,"ON")
15     return render_template("index.html")
16
17 @app.route("/desligarled1", methods = ["POST"])
18 def desligarled1():
19     print("L1 desligada")
20     modificar_estado(1,"OFF")
21     return render_template("index.html")
22
23 @app.route("/ligarled2", methods = ["POST"])
24 def ligarled2():
25     print("L2 ligada")

```

Anexo 4: main.py

```

25     print("L2 ligada")
26     modificar_estado(2, "ON")
27     return render_template("index.html")
28
29 @app.route("/destigarted2", methods = ["POST"])
30 def destigarted2():
31     print("L2 destigada")
32     modificar_estado(2, "OFF")
33     return render_template("index.html")
34
35
36 def modificar_estado(led, estado):
37     f = open('estados.txt', 'r')
38     print(f)
39     for estados in f:
40         print(estados)
41         estados = estados.split(";")
42         print(estados)
43         if estado=="ON":
44             estados[led-1]="1"
45         else:
46             estados[led-1]="0"
47         novos_estados = estados[0] + ";" + estados[1]
48     f.close()

```

Anexo 5: main.py

```

48     f.close()
49     f1 = open('estados.txt', 'w')
50     f1.write(novos_estados)
51     f1.close()
52
53
54
55 if __name__ == "__main__":
56     app.run(debug=True, port = 5000)

```

Anexo 6: main.py

```
host.py venv X
C: > Users > Particular > Desktop > projeto2 > SEIL-pjto2 > venv > host.py
1  import socket
2  import threading
3  import time
4
5  #define o ip do servidor coo endereço IP do HOST
6  SERVER_IP = socket.gethostbyname(socket.gethostname())
7  PORT = 5050
8  ADDR = (SERVER_IP, PORT)
9  #define o formato de comunicação como utf-8
10 FORMATO = 'utf-8'
11
12 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13 server.bind(ADDR)
14
15 conexoes = []
16 mensagens = []
17 estados_leds = ["0", "0"]
18 clock = time.time()
19
20 def enviar_estados(conexao):
21     #Essa função manda a mensagem i da lista mensagens para uma conexao
22     print(f"[ENVIANDO] Enviando mensagens para {conexao['addr']}")
23     mensagem_de_envio = "leds=" + estados_leds[0] + ";" + estados_leds[1]
24     conexao['conn'].send(mensagem_de_envio.encode())
25     time.sleep(0.2)
```

Anexo 7: host.py(socket)

```

25     time.sleep(0.2)
26     print('fim enviar estados')
27
28     """
29     1 vez que o cliente entrar, vai mandar o nome:
30     nome=.....
31     E as mensagens vem:
32     msg=
33     """
34
35     def handle_clientes(conn, addr):
36         #Essa função recebe o endereço, a conexão e o nome de um cliente
37         #e envia uma mensagem individual para ele
38         #ou envia todas as mensagens para os clientes
39         print(f"[NOVA CONEXAO] Um novo usuario se conectou pelo endereço {addr}")
40         global conexoes
41         global mensagens
42         recebido = False
43         while(not recebido):
44             msg = conn.recv(1024).decode(FORMATO)
45             if(msg):
46                 recebido = True
47                 if(msg.startswith("state_request")):
48                     mapa_da_conexao = {
49                         "conn": conn,

```

Anexo 8: host.py(socket)

```
host.py venv X
C: > Users > Particular > Desktop > projeto2 > SEII-pjto2 > venv > host.py
77 mag.startswith("-state_request")):
48     mapa_da_conexao = {
49         "conn": conn,
50         "addr": addr,
51         "last": 1
52     }
53     conexoes.append(mapa_da_conexao)
54     enviar_estados(mapa_da_conexao)
55
56
57
58 def start():
59     global clock
60     global estados_leds
61     # abre o server e estabelece a conexão como thread
62     print("[INICIANDO] Iniciando Socket")
63     server.listen()
64     conn, addr = server.accept()
65     while(True):
66         thread = threading.Thread(target=handle_clientes, args=(conn, addr))
67         thread.start()
68         thread.join()
69         print("[ENVIADO]")
70         if time.time()-clock>0.5:
71             clock = time.time()
72             f = open('estados.txt', 'r')
```

Anexo 9: host.py(socket)

```
72     f = open('estados.txt', 'r')
73     for l in f:
74         estados_leds = l.split(';')
75     f.close()
76
77
78 start()
```

Anexo 10: host.py(socket)

7. BIBLIOGRAFIA

[1] - Roteiro de trabalho da matéria de Sistemas Digitais: “Trabalho Final 2 - Roteiro”. Prof. Dr. Éder Alves de Moura, FEMEC, UFU - Universidade Federal de Uberlândia, 2022-1.

[2] - Roteiro de trabalho da matéria de Sistemas Digitais: “Trabalho Final 2 - IoT”. Prof. Dr. Éder Alves de Moura, FEMEC, UFU - Universidade Federal de Uberlândia, 2022-1.