

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ
ESCOLA POLITÉCNICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

BRUNO NARDONI MOSCHETTA
DANIEL PEREIRA LIMA
DANIELA TAMY YUKI
ISABELLA LUCENA CONCEIÇÃO

RELATÓRIO DE EXPERIÊNCIA CRIATIVA:
BUZZAWAY

CURITIBA
2024

BRUNO NARDONI MOSCHETTA
DANIEL PEREIRA LIMA
DANIELA TAMY YUKI
ISABELLA LUCENA CONCEIÇÃO

RELATÓRIO EXPERIÊNCIA CRIATIVA: BUZZAWAY

Trabalho de Conclusão de Curso
apresentado ao Curso de Graduação em
2024 da Pontifícia Universidade Católica
do Paraná, como requisito parcial à
obtenção do título de bacharel em
Computação.

Orientadores: Prof. Dr. Andrey Cabral
Meira e Dr. Joelton Gotz

CURITIBA

2024

Dedicamos a todos que possibilitaram a
criação da BuzzAway

AGRADECIMENTOS

“Aos professores, o grupo da BuzzAway agradece por todo conhecimento transmitido ao longo das aulas, possibilitando-nos construir um projeto incrível, que proporcionou o nosso desenvolvimento acadêmico. Esperamos que, com essa base, possamos seguir nossos estudos e nos tornar excelentes profissionais. Agradecemos imensamente pelo apoio de vocês no desenvolvimento deste projeto.

Daniela: Aos meus colegas de grupo, muito obrigada por todos os momentos na PUC, jamais serão esquecidos, todos os momentos de desenvolvimento de trabalhos, estudos e diversões foram muito especiais. Assim, como a Isa diz “Eu não quero ser só uma colega, quero levar vocês para a vida”. Eu desejo tudo de melhor para vocês. Assim, eu me despeço do grupo com esse último trabalho entregue, muito obrigada pessoal :)”

Agradecemos também ao Tim Maia por sempre ser um guia nos momentos tristes e difíceis. Ensinando a nós que nem tudo é sobre dinheiro, “*O mundo só será bom no dia que todo o dinheiro acabar, mas que não me falte nenhum enquanto isso não acontece*”.

“O segredo de um grande sucesso está
no trabalho de uma grande equipe”

(Murilo Oliveira)

SUMÁRIO

1INTRODUÇÃO	6
2HARDWARE	7
2.1TÓPICOS MQTT	8
3 . WEB.....	9
4BANCO DE DADOS	17
5INTEGRAÇÃO	18
6 .. TDE.....	19
6.1TDE1	19
6.2TDE2	19
6.3TDE3	20
7CONCLUSÃO	24
REFERÊNCIAS.....	25

<u>Figura 1 - Composição de Hardware</u>	8
<u>Figura 2 - Blueprint Flask</u>	10
<u>Figura 3 - Cadastrar Atuadores</u>	11
<u>Figura 4 - Listar Sensores</u>	11
<u>Figura 5 - Editar Sensores</u>	12
<u>Figura 6 - Remover Sensor</u>	12
<u>Figura 7 - Listar Atuadores</u>	13
<u>Figura 8 - Editor Atuadores</u>	13
<u>Figura 9 - Remover Atuadores</u>	14
<u>Figura 10 - Adicionar Atuadores</u>	14
<u>Figura 11 - Página Inicial</u>	15
<u>Figura 12 - Página do histórico de Sensores</u>	15
<u>Figura 13 - Página de Login Administrador</u>	16
<u>Figura 14 - Página de Cadastro de Usuário</u>	16
<u>Figura 15 - Página de login do Usuário</u>	17
<u>Figura 16 - Banco de dados com usuários cadastrados</u>	17
<u>Figura 17 - Slider CSS</u>	21
<u>Figura 18 - Flask run</u>	23
<u>Figura 19 - Disposição do projeto Login</u>	23
<u>Figura 20 - Sing up</u>	24
<u>Figura 21 - Banco de dados inserido</u>	24
<u>Figura 22 - Logando</u>	25
<u>Figura 23 - Bem vindo!</u>	25

1 INTRODUÇÃO

A disciplina “Experiência Criativa: Criando Soluções Computacionais”, ofertada no 3º período do curso de Ciência da Computação, tem como proposta a construção de protótipos de sensores e atuadores por meio de plataformas eletrônicas, a codificação de sistemas web com arquitetura de software, utilizando requisições e respostas de serviços síncronos e assíncronos, além da construção de banco de dados integrado com a aplicação web. O conteúdo também visa a integração do ambiente web com bancos de dados e microcontroladores, utilizando interfaces para solucionar problemas relacionados à Internet das Coisas (IoT).

Com o objetivo de criar soluções inteligentes para a resolução de problemas reais, a equipe debateu e percebeu a necessidade de criar algo para a proteção contra mosquitos transmissores de doenças, que fosse acessível, sem causar impactos ambientais e de baixa manutenção. Foi observado que o mercado oferece principalmente repelentes e dispositivos de veneno, que exigem a compra constante do produto.

Assim, o grupo desenvolveu durante o semestre o projeto BuzzAway, com o objetivo de alertar os usuários sobre a presença de mosquitos no ambiente, com um baixo custo, sem impactos negativos para a natureza e baixa manutenção. O protótipo foi desenvolvido no Wokwi, implementado na web por meio do Flask e integrado com um banco de dados.

2 HARDWARE

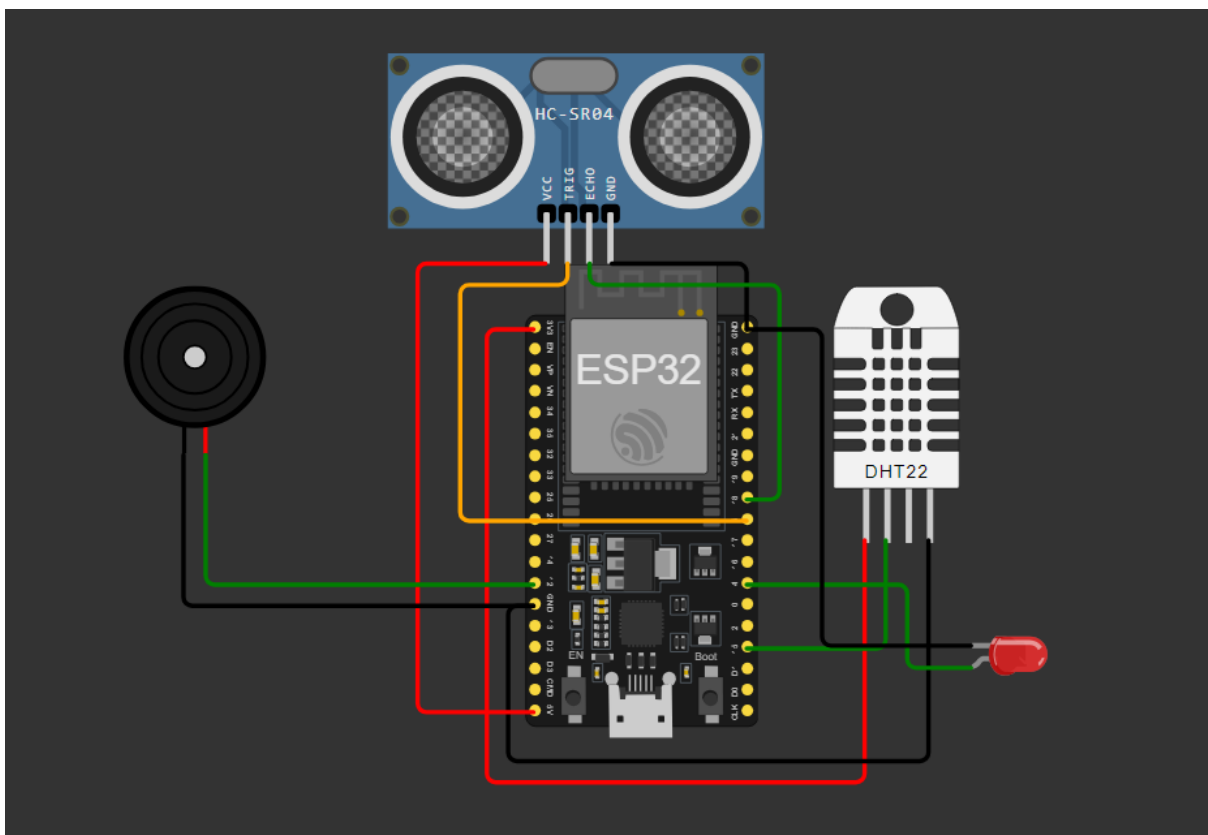


Figura 1 - Composição de Hardware

Na esquemática é apresentada toda a parte do hardware utilizado no projeto. O componente central é o microcontrolador ESP32, famoso pelo seu baixo custo e baixo consumo de energia com Wi-fi integrado e Bluetooth.

À esquerda da ESP, é possível ver um Buzzer ou sensor Piezo, que é um componente eletrônico que pode gerar sons, bips e até mesmo tocar música, no projeto ele funciona como um dos atuadores.

Ao topo tem-se o HC-SR04, um sensor ultrassônico, onde uma das saídas é responsável por transmitir um som imperceptível a audição, que com as métricas corretas e adaptação dos valores é possível se descobrir a distância de um objeto, a outra saída fica constantemente recebendo esse “som” que possibilita a descoberta da distância, como o próprio nome já diz, funcionará como um sensor.

À direita da ESP, encontra-se o sensor de temperatura e umidade, o DHT22, que usa um termistor para medir a temperatura e um sensor capacitivo para medir umidade. Novamente, como o nome diz, ambos funcionarão como sensores, totalizando três.

Por último e não menos importante, ao lado do DHT tem também um led vermelho que funcionará como um sinalizador.

2.1 TÓPICOS MQTT

O MQTT é um protocolo de mensagens que permite a comunicação entre dispositivos em uma rede. Cada dado é publicado em um tópico específico, assim, os dispositivos se inscrevem em seus respectivos tópicos, o que faz com que recebam todas as mensagens publicadas no mesmo. As mensagens podem ter vários significados e intuitos.

No caso do projeto do BuzzAway, existem 6 tópicos no total.

MQTT_TOPIC1 = “Umidade/Computistas”: Este tópico é usado para publicar os dados de umidade coletados pelo sensor DHT22.

MQTT_TOPIC2 = “Temperatura/Computistas”: Este tópico é usado para publicar os dados de temperatura coletados pelo sensor DHT22.

MQTT_TOPIC3 = “SensorUltrassom/Computistas”: Este tópico é usado para publicar os dados de distância coletados pelo sensor ultrassônico.

MQTT_TOPIC4 = “Buzzer/Computistas”: Este tópico é usado para publicar o status do buzzer (ligado/desligado).

MQTT_TOPIC5 = “SituacaoAmbiente/Computistas”: Este tópico é usado para publicar a situação do ambiente com base nas condições de temperatura e umidade.

MQTT_TOPIC6 = “LedMosquito/Computistas”: Este tópico é usado para publicar o status do LED (ligado/desligado).

3 WEB

Para o desenvolvimento web neste projeto, foi utilizado o framework flask para a conexão entre HTML, CSS e entre outros. Além disso, foi feita uma função do Flask denominada Flask-Login, o qual pode ser usada para login de usuários no site via flask.

Na aplicação, foi realizada com 3 blueprints (imagem X), para organizar e estruturar grandes projetos. Além disso, esse método, possibilita a modularização do código, facilitando a manutenção e escalabilidade. Assim, no projeto BuzzAway houve a implementação de blueprints para sensores, atuadores e para o read.

```
app.register_blueprint(sensors, url_prefix='/')  
app.register_blueprint(actuator, url_prefix='/')  
app.register_blueprint(read, url_prefix='/')
```

Figura 2 - Blueprint Flask

Para cadastrar os sensores e atuadores utilizamos o Flask junto com o banco de dados, recebendo informações relacionadas ao nome, marca, modelo, unidade de medida utilizada, o tópico mqtt e o status de funcionamento. Com esses elementos todos devidamente preenchidos, é possível carregar os dados no banco e exibir os dados cadastrados na página de listagem. Ademais, na página que as informações são mostradas, o administrador tem o poder de editar e deletar sensores e atuadores que foram cadastrados incorretamente ou que não são mais desejados.

Imagem 2 - Cadastrar sensores

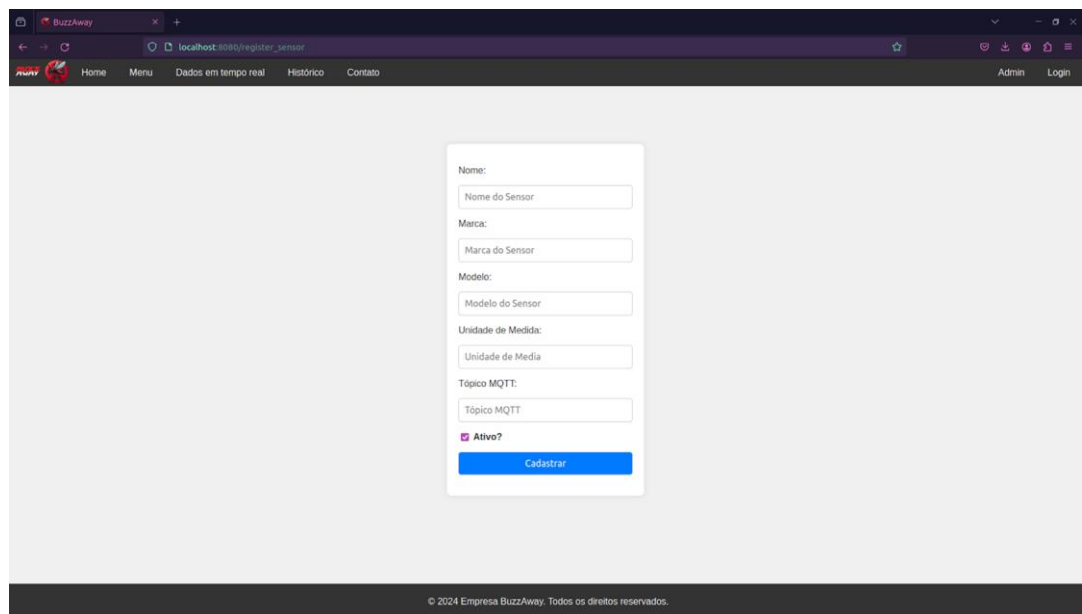


Figura 3 - Cadastrar Atuadores

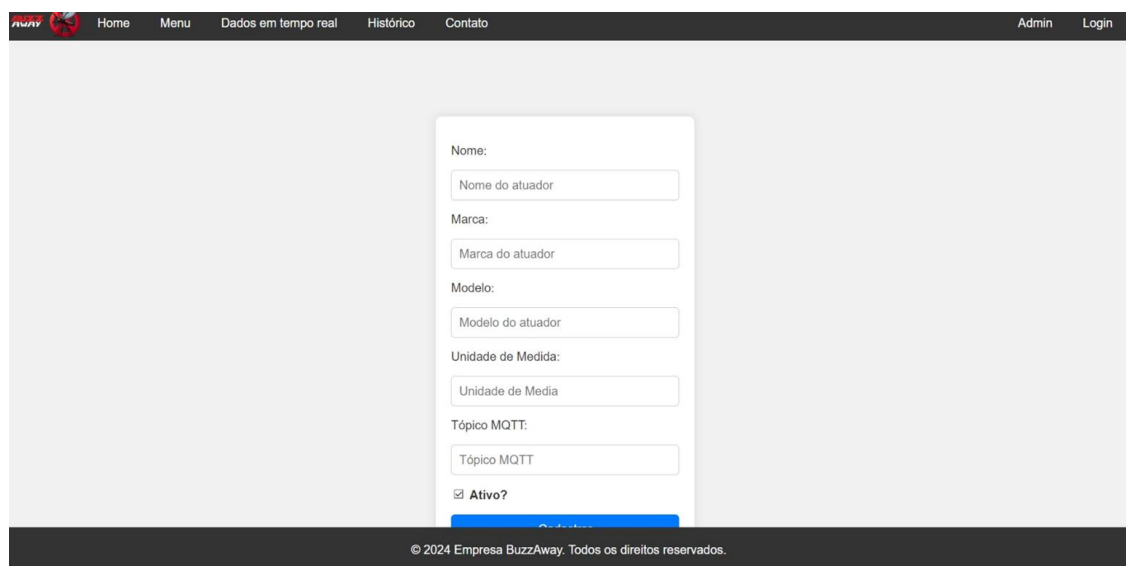


Figura 4 - Listar Sensores

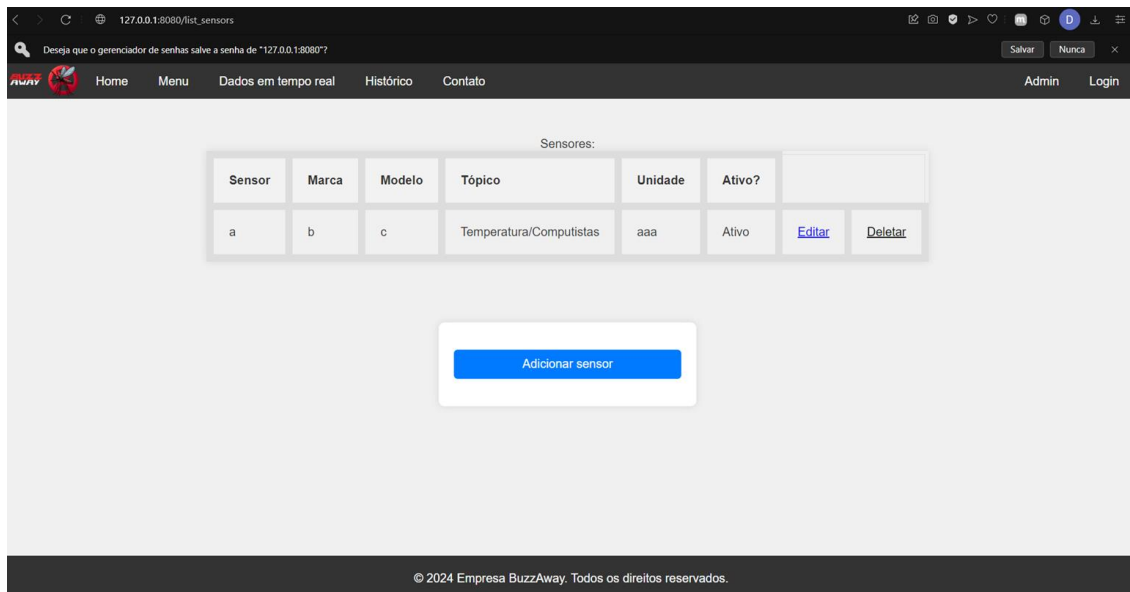


Figura 5 - Editar Sensores

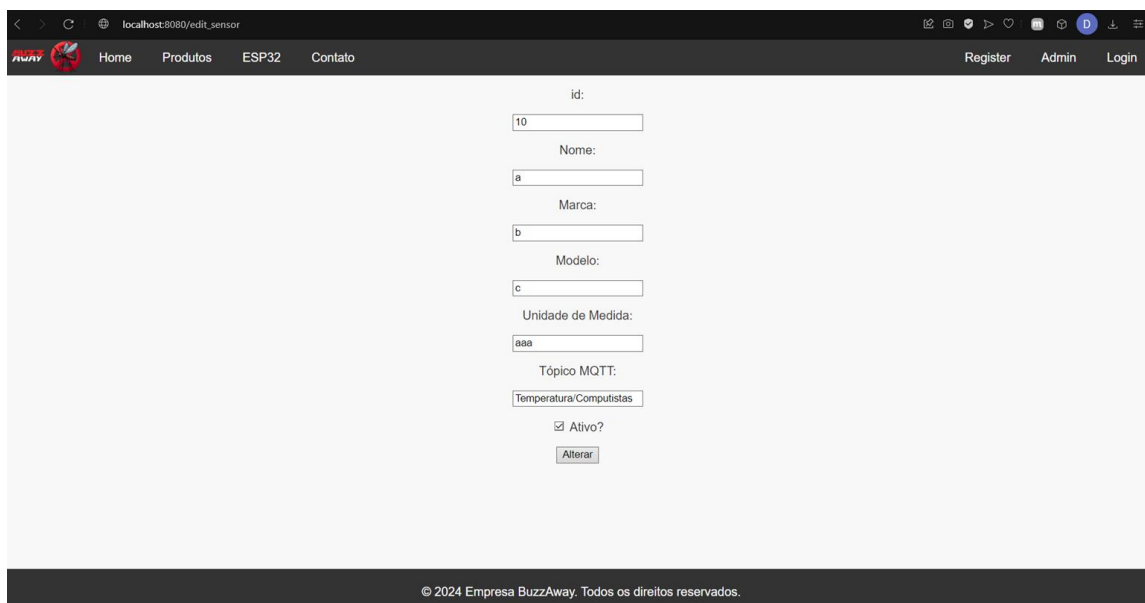


Figura 6 - Remover Sensor

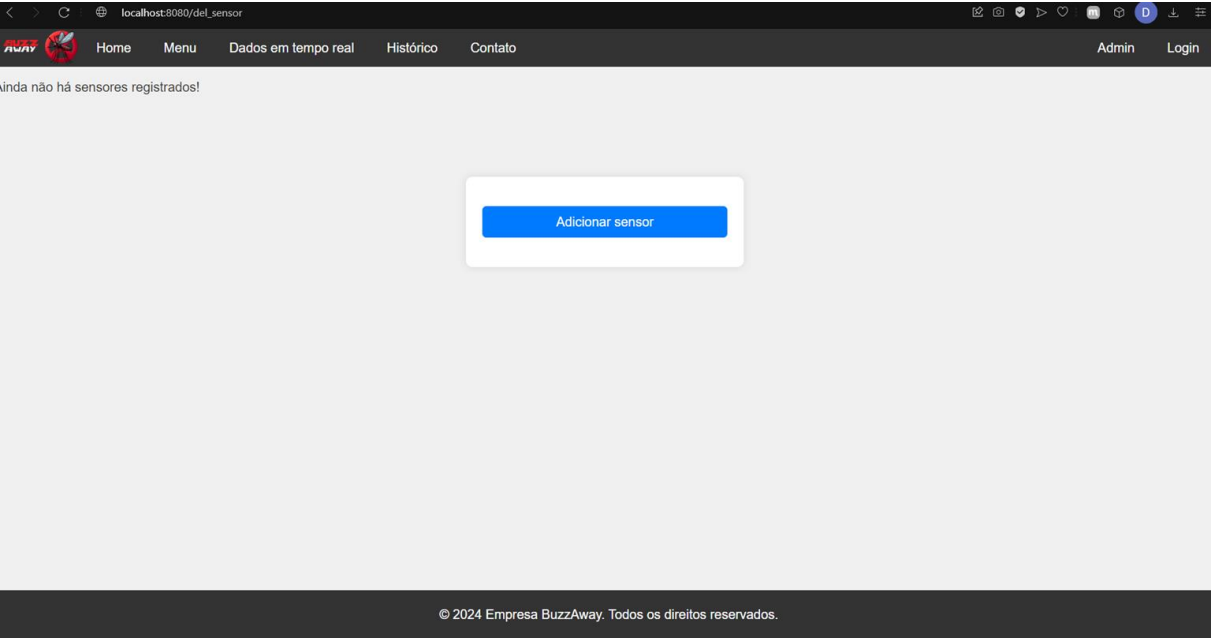


Figura 7 - Listar Atuadores

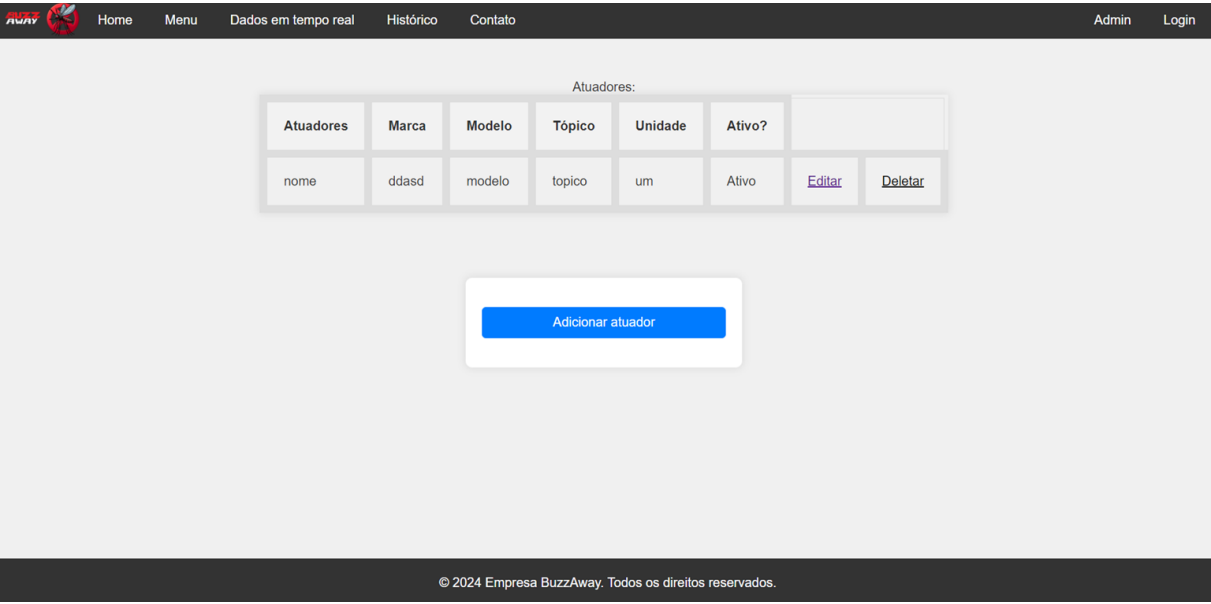
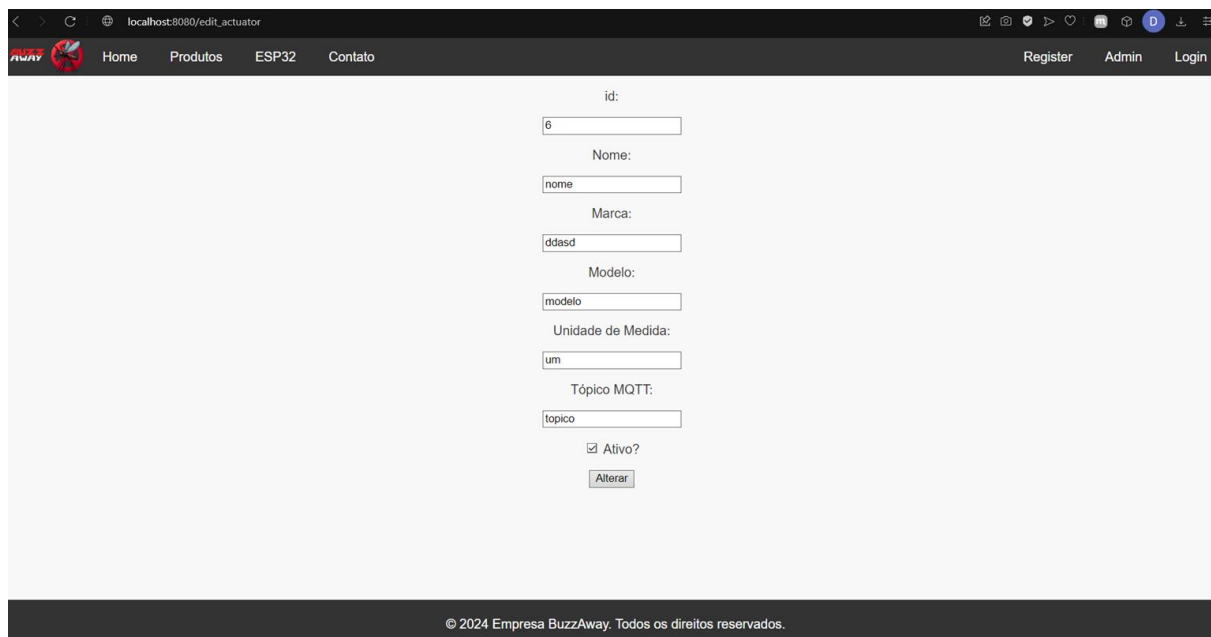


Figura 8 - Editor Atuadores



id:

6

Nome:

nome

Marca:

ddasd

Modelo:

modelo

Unidade de Medida:

um

Tópico MQTT:

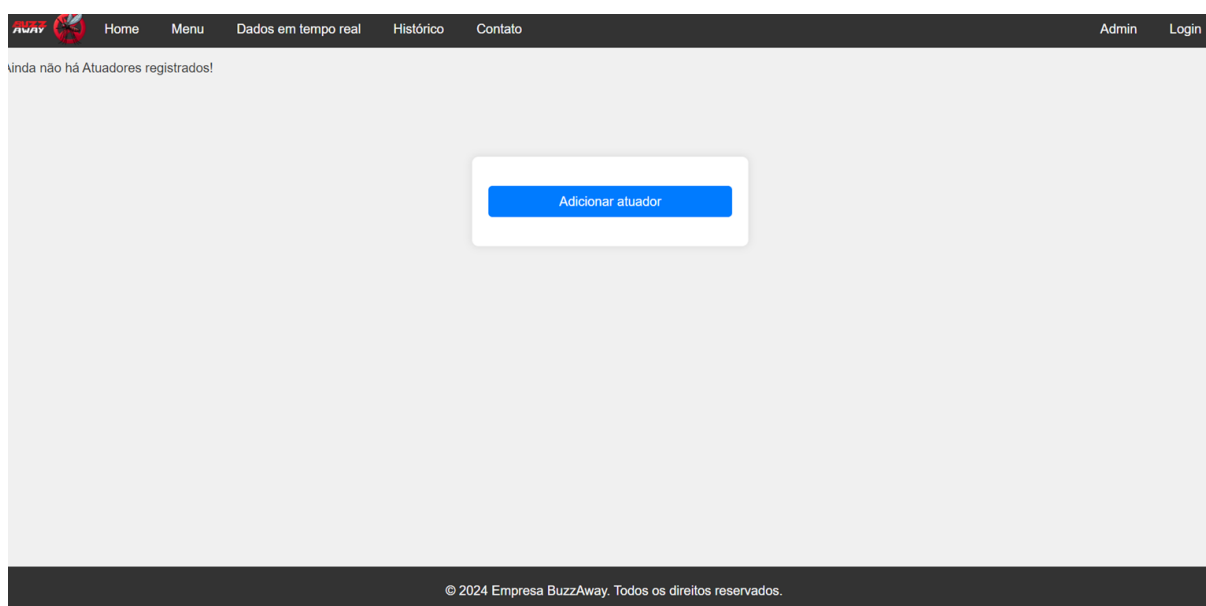
topico

☒ Ativo?

Alterar

© 2024 Empresa BuzzAway. Todos os direitos reservados.

Figura 9 - Remover Atuadores



Ainda não há Atuadores registrados!

Adicionar atuador

© 2024 Empresa BuzzAway. Todos os direitos reservados.

Figura 10 - Adicionar Atuadores

A página inicial do site traz uma mensagem de alerta: “A dengue mata. Dê um Buzz”, fazendo referência ao objetivo do projeto. Outrossim, essa página mostra a representação física de como ficaria o produto final feito com inteligência artificial pelo Copilot.

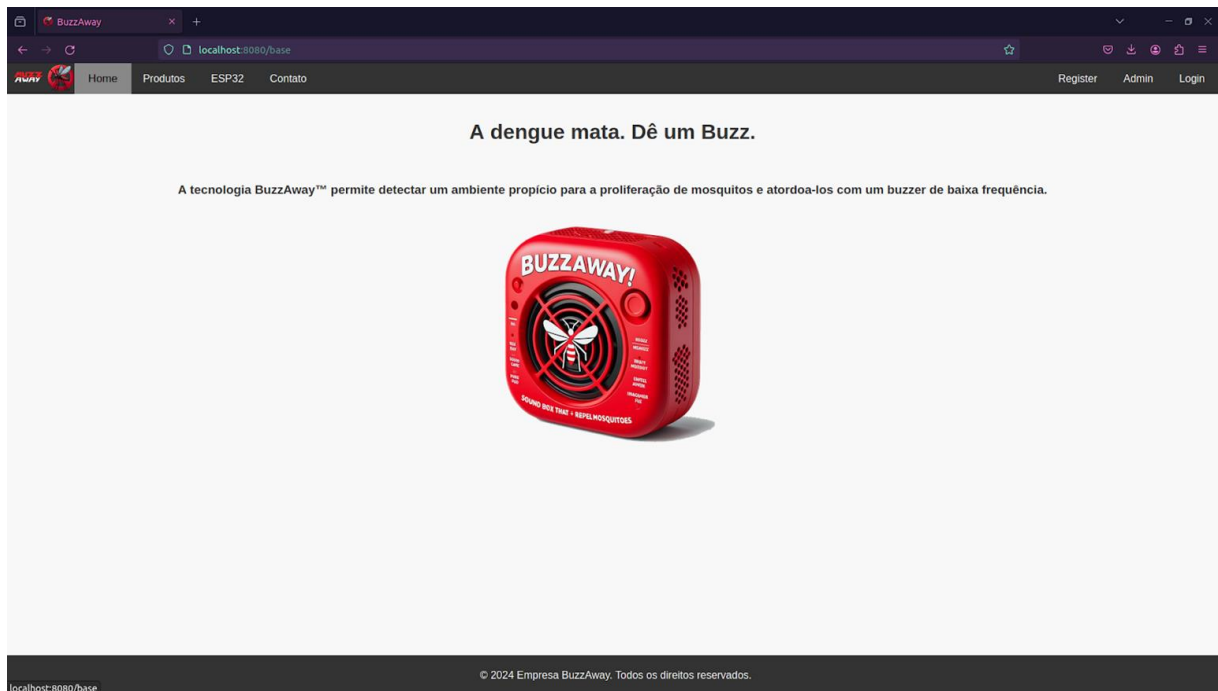


Figura 11 - Página Inicial

As páginas do histórico de sensores e atuadores são responsáveis por retornar os dados em um determinado período fornecido pelo usuário administrador, de acordo com o sensor ou atuador cadastrado anteriormente.

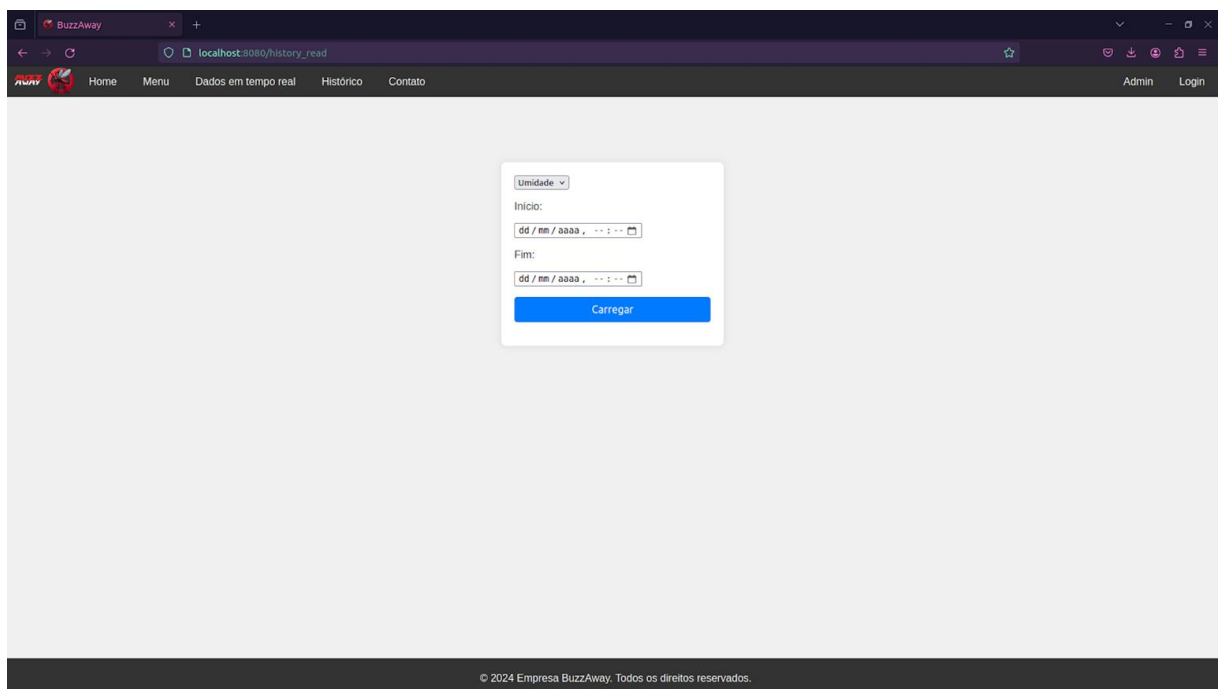
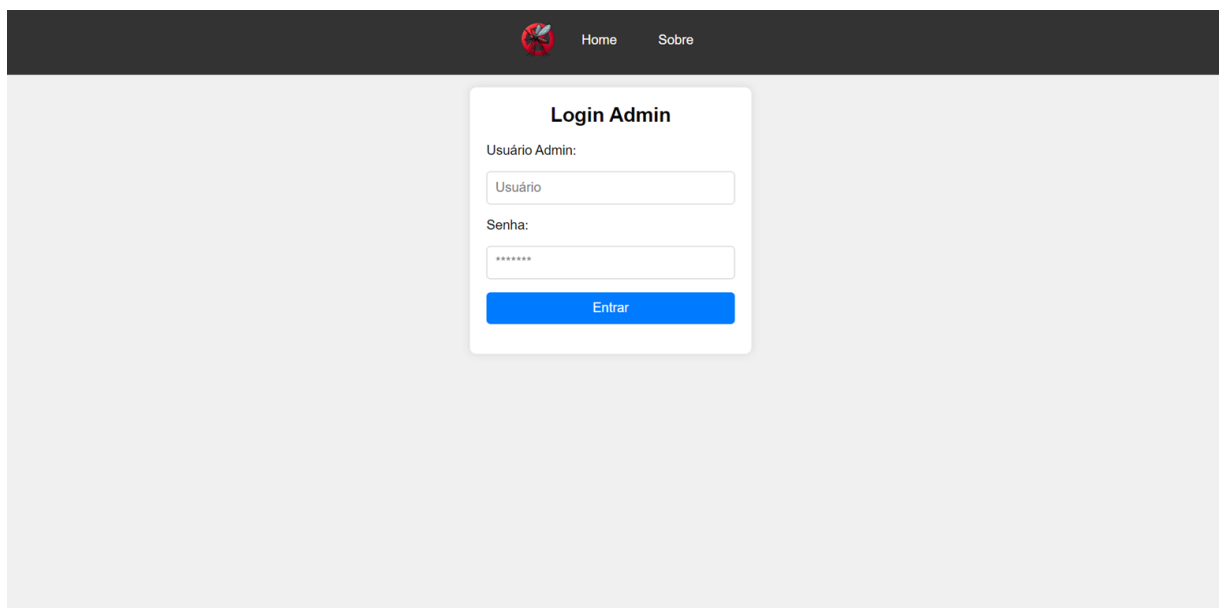


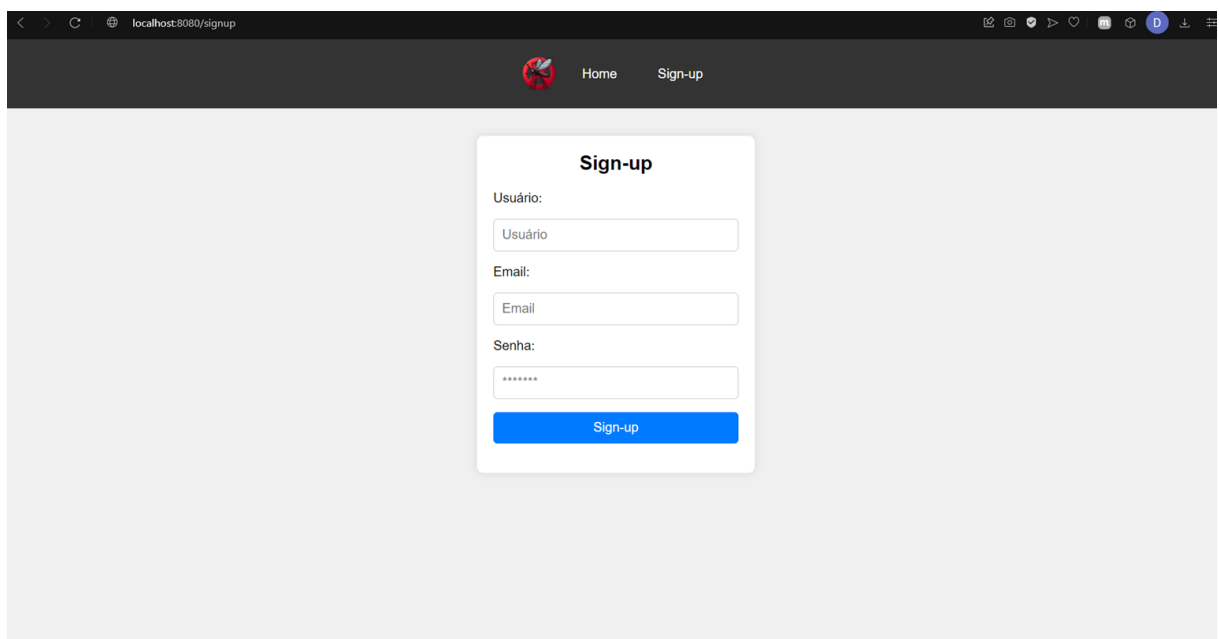
Figura 12 - Página do histórico de Sensores

Para realizar o login foi utilizado o Flask-Login junto com banco de dados, o usuário realiza o seu cadastro no site na página “Sign Up” e depois pode realizar o login, com o email e senha cadastrados anteriormente. Ademais, a senha no banco é armazenada com criptografia, protegendo os dados dos clientes. A página de entrada de administrador fica separada, apenas o administrador pode executar o cadastro e edição.



The screenshot shows a web application interface with a dark header bar containing a logo and navigation links for 'Home' and 'Sobre'. The main content area is light gray and features a white 'Login Admin' form. The form includes a label 'Usuário Admin:', a text input field for 'Usuário', a label 'Senha:', a password input field with masked characters, and a blue 'Entrar' button.

Figura 13 - Página de Login Administrador



The screenshot shows a web application interface with a dark header bar containing a logo and navigation links for 'Home' and 'Sign-up'. The main content area is light gray and features a white 'Sign-up' form. The form includes labels for 'Usuário:', 'Email:', and 'Senha:', each followed by a corresponding text input field. The password field is masked with asterisks. A blue 'Sign-up' button is at the bottom of the form.

Figura 14 - Página de Cadastro de Usuário

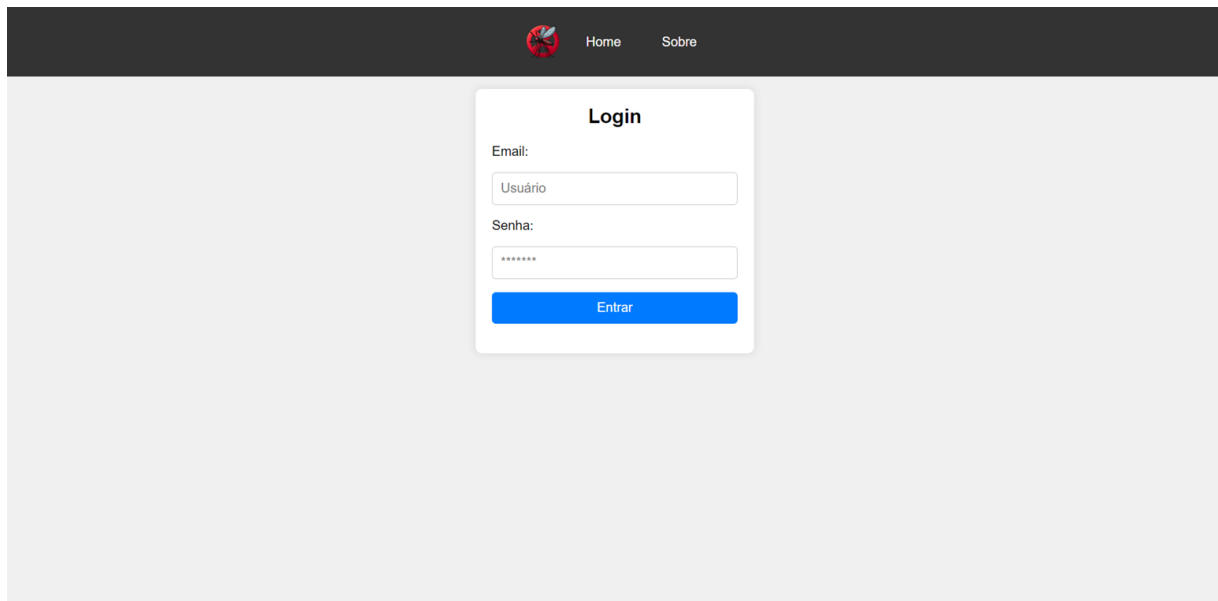


Figura 15 - Página de login do Usuário

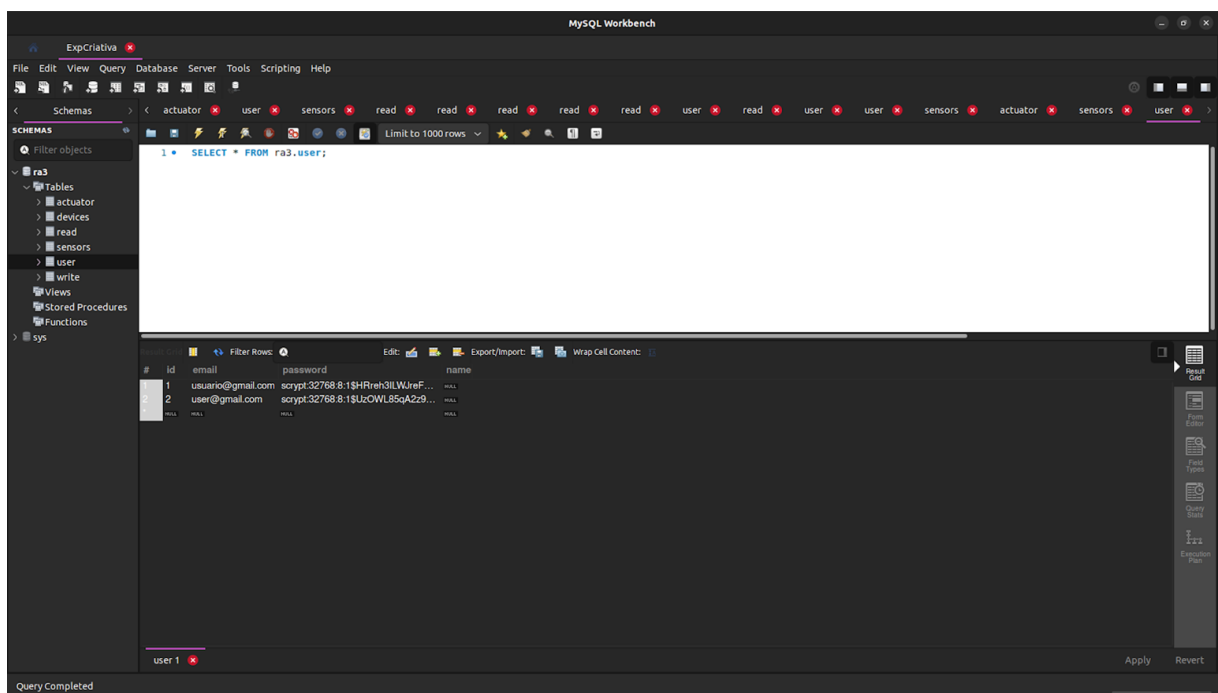


Figura 16 - Banco de dados com usuários cadastrados

4 BANCO DE DADOS

Foi usado para o banco de dados, o Flask juntamente com SQLAlchemy e SQLWorkbench. Para podermos usar eles, devemos instanciar métodos para as tabelas serem criadas dentro do SQLWorkbench usando o SQLAlchemy. Para sensores e atuadores deve se fazer o mesmo, porém também métodos para receber os tópicos e entradas do usuário pelo site. A tabela Usuário foi criado da mesma forma, porém usando FlaskLogin, para poder encriptar sua senha quando se registrar na plataforma. A seguir as imagens tanto do flask, SQLAlchemy e SQLWorkbench.

A criação das tabelas pelo flask e SQLAlchemy:

Tabela de dispositivos:

```
models > iot > devices.py > ...
1  from models.db import db
2  class Device(db.Model):
3      __tablename__ = 'devices'
4      id= db.Column('id', db.Integer, primary_key=True)
5      name= db.Column(db.String(50))
6      brand= db.Column(db.String(50))
7      model= db.Column(db.String(50))
8      is_active= db.Column(db.Boolean, nullable= False, default= False)
9      sensors = db.relationship('Sensor', backref='devices', lazy=True)
10     actuator = db.relationship('Actuator', backref='devices', lazy=True)
11
12
```

Atuadores:

```

models > iot > actuator.py > ...
1  from models.db import db
2  from models.iot.devices import Device
3  class Actuator(db.Model):
4      __tablename__ = 'actuator'
5      id= db.Column('id', db.Integer, primary_key=True)
6      devices_id = db.Column( db.Integer, db.ForeignKey(Device.id))
7      unit = db.Column(db.String(50))
8      topic = db.Column(db.String(50))
9
models > iot > actuator.py > ...
3  class Actuator(db.Model):
9
10     def save_actuator(name, brand, model, topic, unit, is_active):
11         device = Device(name = name, brand = brand,
12             model = model, is_active = is_active)
13         actuator = Actuator(devices_id = device.id, unit= unit, topic = topic)
14         device.actuator.append(actuator)
15         db.session.add(device)
16         db.session.commit()
17
18     def get_actuator():
19         actuator = Actuator.query.join(Device, Device.id == Actuator.devices_id)\
20             .add_columns(Device.id, Device.name,
21                 Device.brand, Device.model,
22                 Device.is_active, Actuator.topic,
23                 Actuator.unit).all()
24         return actuator
25
26     def get_single_actuator(id):
27         actuator = Actuator.query.filter(Actuator.devices_id == id).first()
28         if actuator is not None:
29             actuator = Actuator.query.filter(Actuator.devices_id == id)\
30                 .join(Device).add_columns(Device.id, Device.name, Device.brand,
31                     Device.model, Device.is_active, Actuator.topic, Actuator.unit).first()
32         return [actuator]
33
34     def update_actuator(id,name, brand, model, topic, unit, is_active):
35         device = Device.query.filter(Device.id == id).first()
36         actuator = Actuator.query.filter(Actuator.devices_id == id).first()
37         if device is not None:
38             device.name = name
39             device.brand = brand
40             device.model = model
41             actuator.topic = topic
42             actuator.unit = unit
43             device.is_active = is_active
44             db.session.commit()
45             return Actuator.get_actuator()
46
47     def delete_actuator(id):
48         device = Device.query.filter(Device.id == id).first()
49         actuator = Actuator.query.filter(Actuator.devices_id == id).first()
50         db.session.delete(actuator)
51         db.session.delete(device)
52         db.session.commit()
53         return Actuator.get_actuator()
54

```

Sensores:

```

models > iot > sensors.py > Sensor > get_single_sensor
1  from models.db import db
2  from models.iot.devices import Device
3  class Sensor(db.Model):
4      __tablename__ = 'sensors'
5      id= db.Column('id', db.Integer, primary_key=True)
6      devices_id = db.Column( db.Integer, db.ForeignKey(Device.id))
7      unit = db.Column(db.String(50))
8      topic = db.Column(db.String(50))
9
models > iot > sensors.py > Sensor > get_single_sensor
3  class Sensor(db.Model):
9
10     def save_sensor(name, brand, model, topic, unit, is_active):
11         device = Device(name = name, brand = brand,
12             model = model, is_active = is_active)
13         sensor = Sensor(devices_id = device.id, unit= unit, topic = topic)
14         device.sensors.append(sensor)
15         db.session.add(device)
16         db.session.commit()
17
18     def get_sensors():
19         sensors = Sensor.query.join(Device, Device.id == Sensor.devices_id)\
20             .add_columns(Device.id, Device.name,
21                 Device.brand, Device.model,
22                 Device.is_active, Sensor.topic,
23                 Sensor.unit).all()
24         return sensors
25
26     def get_single_sensor(id):
27         sensor = Sensor.query.filter(Sensor.devices_id == id).first()
28         if sensor is not None:
29             sensor = Sensor.query.filter(Sensor.devices_id == id)\
30                 .join(Device).add_columns(Device.id, Device.name, Device.brand,
31                     Device.model, Device.is_active, Sensor.topic, Sensor.unit).first()
32         return [sensor]
33
34     def update_sensor(id,name, brand, model, topic, unit, is_active):
35         device = Device.query.filter(Device.id == id).first()
36         sensor = Sensor.query.filter(Sensor.devices_id == id).first()
37         if device is not None:
38             device.name = name
39             device.brand = brand
40             device.model = model
41             sensor.topic = topic
42             sensor.unit = unit
43             device.is_active = is_active
44             db.session.commit()
45             return Sensor.get_sensors()
46
47     def delete_sensor(id):
48         device = Device.query.filter(Device.id == id).first()
49         sensor = Sensor.query.filter(Sensor.devices_id == id).first()
50         db.session.delete(sensor)
51         db.session.delete(device)
52         db.session.commit()
53         return Sensor.get_sensors()
54

```

Write:

```
models > iot > write.py > ...
1  from models.db import db
2  from models.iot.actuator import Actuator
3  from datetime import datetime
4  from models.iot.devices import Device
5
6  class Write(db.Model):
7      tablename = 'write'
8      id= db.Column('id', db.Integer, nullable = False, primary_key=True)
9      write_datetime = db.Column(db.DateTime(), nullable = False)
10     actuator_id= db.Column(db.Integer, db.ForeignKey(Actuator.id), nullable = False)
11     value = db.Column( db.Float, nullable = True)
12
13     def save_write(topic, value):
14         actuator = Actuator.query.filter(Actuator.topic == topic).first()
15         device = Device.query.filter(Device.id == actuator.devices_id).first()
16
17         if (actuator is not None) and (device.is_active==True):
18             print(topic, value)
19             print(actuator)
20             print(device)
21             write = Write( write_datetime = datetime.now(), actuator_id = actuator.id, value = float(value) )
22             db.session.add(write)
23             db.session.commit()
24
25     def get_write(actuator_id, start, end):
26         write = Write.query.filter(Write.actuator_id == actuator_id,
27                                   Write.write_datetime > start,
28                                   Write.write_datetime<end).all()
29         return write
```

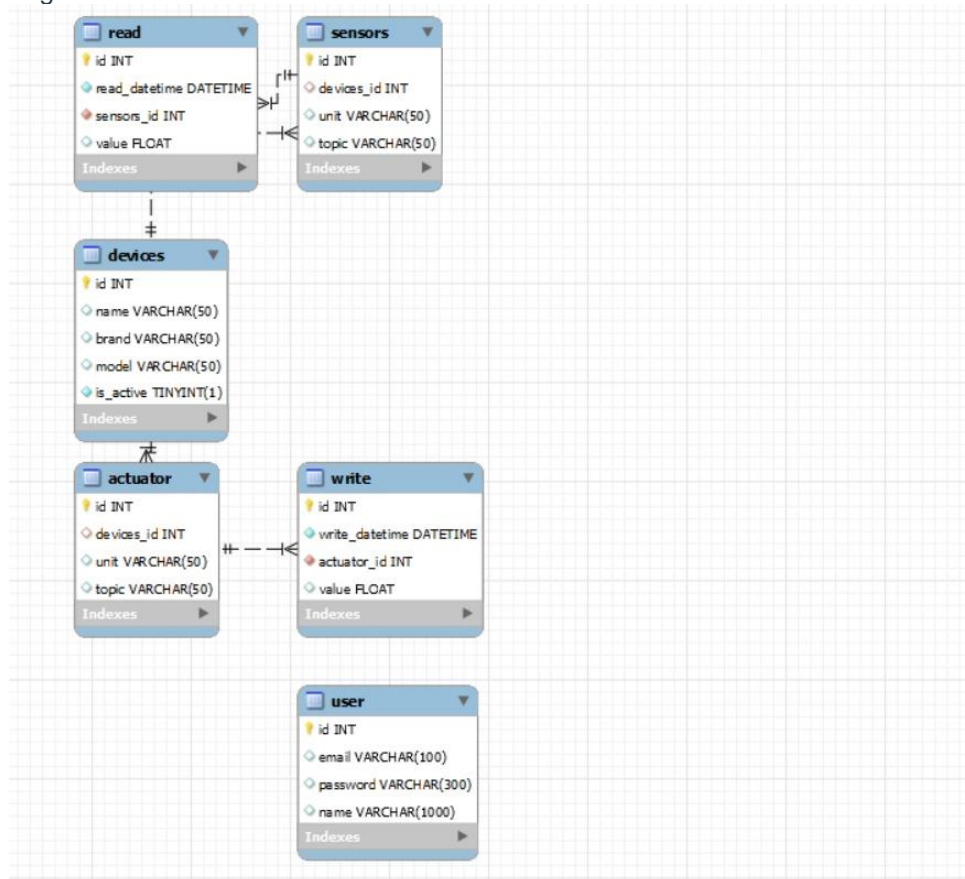
Read:

```
models > iot > read.py > ...
1  from models.db import db
2  from models.iot.sensors import Sensor
3  from models.iot.actuator import Actuator
4  from models.iot.devices import Device
5  from datetime import datetime
6  class Read(db.Model):
7      tablename = 'read'
8      id= db.Column('id', db.Integer, nullable = False, primary_key=True)
9      read_datetime = db.Column(db.DateTime(), nullable = False)
10     sensors_id= db.Column(db.Integer, db.ForeignKey(Sensor.id), nullable = False)
11     value = db.Column( db.Float, nullable = True)
12
13     def save_read(topic, value):
14         sensor = Sensor.query.filter(Sensor.topic == topic).first()
15         device = Device.query.filter(Device.id == sensor.devices_id).first()
16         if (sensor is not None) and (device.is_active==True):
17             read = Read( read_datetime = datetime.now(), sensors_id = sensor.id, value = float(value) )
18             db.session.add(read)
19             db.session.commit()
20
21     def get_read(device_id, start, end):
22         sensor = Sensor.query.filter(Sensor.devices_id == device_id).first()
23         read = Read.query.filter(Read.sensors_id == sensor.id,
24                                 Read.read_datetime > start,
25                                 Read.read_datetime<end).all()
26         return read
```

User:

```
models > user.py > ...
1  from flask_login import UserMixin
2  from .db import db
3
4
5  class User(UserMixin, db.Model):
6      id = db.Column(db.Integer, primary_key=True) # primary keys are required by SQLAlchemy
7      email = db.Column(db.String(100), unique=True)
8      password = db.Column(db.VARCHAR(300))
9      name = db.Column(db.String(1000))
```

Modelo Lógico do banco de dados:



Tabelas e colunas no SQLWorkbench:

Table	Column	Type	Default Value	Nullable	Character	Collation	Privileges	Extra	Comments
actuator	id	int		NO			select,insert,update,reference	auto_increment	
actuator	devices_id	int		YES			select,insert,update,reference		
actuator	unit	varchar(50)		YES	utf8mb4	utf8mb4_0900	select,insert,update,reference		
actuator	topic	varchar(50)		YES	utf8mb4	utf8mb4_0900	select,insert,update,reference		
devices	id	int		NO			select,insert,update,reference	auto_increment	
devices	name	varchar(50)		YES	utf8mb4	utf8mb4_0900	select,insert,update,reference		
devices	brand	varchar(50)		YES	utf8mb4	utf8mb4_0900	select,insert,update,reference		
devices	model	varchar(50)		YES	utf8mb4	utf8mb4_0900	select,insert,update,reference		
devices	is_active	tinyint(1)		NO			select,insert,update,reference		
read	id	int		NO			select,insert,update,reference	auto_increment	
read	read_datetime	datetime		NO			select,insert,update,reference		
read	sensors_id	int		NO			select,insert,update,reference		
read	value	float		YES			select,insert,update,reference		
sensors	id	int		NO			select,insert,update,reference	auto_increment	
sensors	devices_id	int		YES			select,insert,update,reference		
sensors	unit	varchar(50)		YES	utf8mb4	utf8mb4_0900	select,insert,update,reference		
sensors	topic	varchar(50)		YES	utf8mb4	utf8mb4_0900	select,insert,update,reference		
user	id	int		NO			select,insert,update,reference	auto_increment	
user	email	varchar(100)		YES	utf8mb4	utf8mb4_0900	select,insert,update,reference		
user	password	varchar(300)		YES	utf8mb4	utf8mb4_0900	select,insert,update,reference		
user	name	varchar(1000)		YES	utf8mb4	utf8mb4_0900	select,insert,update,reference		
write	id	int		NO			select,insert,update,reference	auto_increment	
write	write_datetime	datetime		NO			select,insert,update,reference		
write	actuator_id	int		NO			select,insert,update,reference		
write	value	float		YES			select,insert,update,reference		

Criação da data base pelo Flask:


```
from flask import Flask
from models import *

def create_db(app:Flask):
    with app.app_context():
        # db.drop_all()
        # db.create_all()
        pass
```

Instância na Main:

```
# app.py
from controllers.app_controller import create_app
from utils.create_db import create_db
if __name__ == "__main__":
    app = create_app()
    create_db(app)
    app.run(host='0.0.0.0', port=8080, debug=False)
```

5 INTEGRAÇÃO

```
def create_app():

    app = Flask(__name__,
                template_folder="./templates/",
                static_folder="./static/",
                root_path=".")

    app.config['TESTING'] = False
    app.config['SECRET_KEY'] = 'generated-secrete-key'
    app.config['SQLALCHEMY_DATABASE_URI'] = instance

    db.init_app(app)

    login_manager = LoginManager()
    login_manager.login_view = 'login'
    login_manager.init_app(app)

    app.config['MQTT_BROKER_URL'] = 'mqtt-dashboard.com'
    app.config['MQTT_BROKER_PORT'] = 1883
    app.config['MQTT_USERNAME'] = '' # Set this item when you need to verify username and password
    app.config['MQTT_PASSWORD'] = '' # Set this item when you need to verify username and password
    app.config['MQTT_KEEPALIVE'] = 5000 # Set KeepAlive time in seconds
    app.config['MQTT_TLS_ENABLED'] = False # If your broker supports TLS, set it True

    mqtt_client = Mqtt()
    mqtt_client.init_app(app)

    topic_subscribe1 = "Umidade/Computistas"
    topic_subscribe2 = "Temperatura/Computistas"
    topic_subscribe3 = "SensorUltrassom/Computistas"
    topic_subscribe4 = "Buzzer/Computistas"
    topic_subscribe5 = "SituBuzzer/ComputistasacaoAmbiente/Computistas"
    topic_subscribe6 = "LedMosquito/Computistas"

    @mqtt_client.on_connect()
    def handle_connect(client, userdata, flags, rc):
        if rc == 0:
            print('Broker Connected successfully')
```

```
def create_app():

    @mqtt_client.on_connect()
    def handle_connect(client, userdata, flags, rc):
        if rc == 0:
            print('Broker Connected successfully')
            mqtt_client.subscribe(topic_subscribe1)
            mqtt_client.subscribe(topic_subscribe2)
            mqtt_client.subscribe(topic_subscribe3)
            mqtt_client.subscribe(topic_subscribe4)
            mqtt_client.subscribe(topic_subscribe5)
            mqtt_client.subscribe(topic_subscribe6) # subscribe topic
        else:
            print('Bad connection. Code:', rc)

    @mqtt_client.on_disconnect()
    def handle_disconnect(client, userdata, rc):
        print("Disconnected from broker")

    @mqtt_client.on_message()
    def handle_mqtt_message(client, userdata, message):
        global temperature, humidity, ultrassom
        if(message.topic==topic_subscribe1):
            js = json.loads(message.payload.decode())
            temperature = js
            try:
                with app.app_context():
                    Read.save_read(topic_subscribe1, temperature)
            except:
                pass
        elif(message.topic==topic_subscribe2):
            js = json.loads(message.payload.decode())
            humidity = js
            try:
                with app.app_context():
                    Read.save_read(topic_subscribe2, humidity)
            except:
                pass
        elif(message.topic==topic_subscribe3):
            js = json.loads(message.payload.decode())
            ultrassom = js
        elif(message.topic==topic_subscribe4):
```

O banco de dados foi configurado para armazenar todos os dados coletados pelos sensores e os comandos enviados para os atuadores. Utilizamos o Flask junto com o SQLAlchemy para gerenciar as tabelas no banco de dados, que incluem registros de sensores, atuadores e usuários. Isso permitiu uma fácil integração e consulta de dados dentro da aplicação web, garantindo que todas as informações fossem salvas de maneira organizada e acessível para futuras análises. (Conforme mostrado em 4. Banco de dados)

Para o desenvolvimento web, utilizamos o Flask devido à sua simplicidade e flexibilidade. Organizamos o código com blueprints para facilitar a manutenção e escalabilidade. A comunicação IoT foi integrada

usando Flask-MQTT, permitindo a recepção de dados dos sensores e o envio de comandos aos atuadores em tempo real. Também implementamos templates HTML e CSS para criar uma interface de usuário funcional e agradável, além é claro, do JavaScript para aplicações visuais e dinâmicas do projeto.

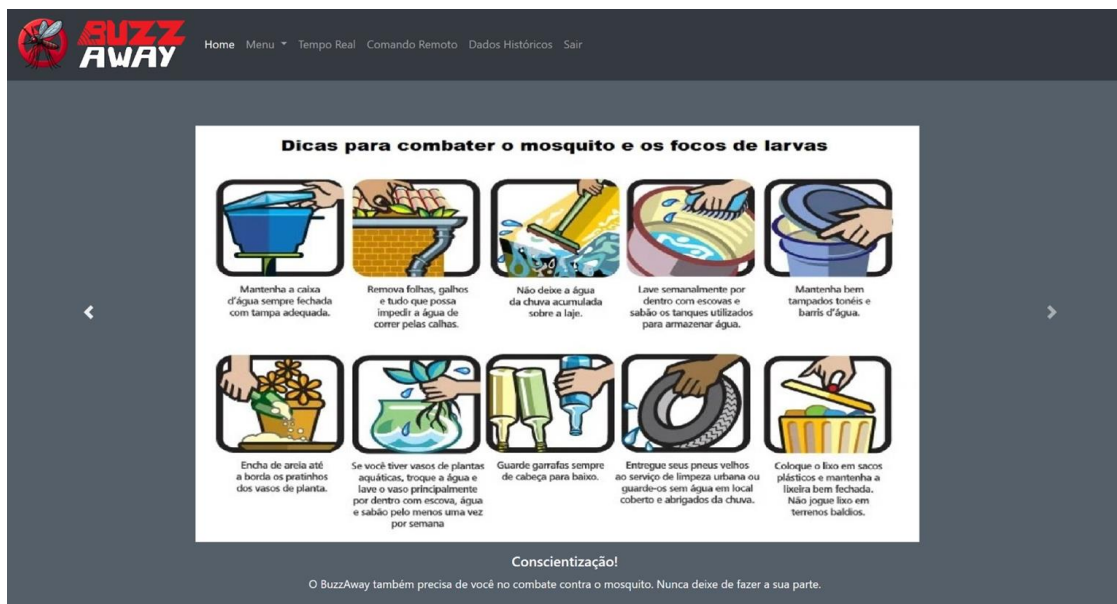
```
document.addEventListener("DOMContentLoaded", function() {  
  ⚡ var form = document.getElementById("loginForm");  
  form.style.opacity = 0;  
  form.style.transform = "translateY(20px)";  
  
  setTimeout(function() {  
    form.style.transition = "opacity 2s ease, transform 2s ease";  
    form.style.opacity = 1;  
    form.style.transform = "translateY(0)";  
  }, 100);  
});
```

6 TDE

Conforme o solicitado pelos orientadores, o grupo realizou os trabalhos discentes efetivos.

6.1 TDE1

Em algum ponto no início do projeto, após as implementações iniciais do site que já constavam com JavaScript, decidimos usar o framework Front-End Bootstrap, para aplicar funções como um slider interativo, minimalista e moderno, pensando em uma interação dinâmica com o usuário.



Portanto, devido a problemas de compatibilidade com o JavaScript já aplicado optamos por utilizar todos os recursos Front-End do Flask e deixar o resto independente de frameworks externos, finalizando o projeto com recursos front-end aplicados pelo nosso próprio código JavaScript, pensando em um projeto mais maleável, com facilidade de manutenção. O grupo decidiu que, depender de mais um framework para o projeto poderia aumentar a vulnerabilidade do projeto a problemas de segurança, pois qualquer falha em um desses frameworks pode comprometer a integridade de toda a aplicação. Portanto, manter o uso de frameworks ao mínimo necessário pode ajudar a manter o projeto mais ágil, seguro e fácil de manter a longo

prazo. Com o JavaScript aplicado, construímos uma interface elegante, moderna com controle de opacidade e movimento de formulários.

6.2 TDE2

Na atividade “TDE2 – criação de aplicação IOT no Wokwi” foi desenvolvido a aplicação para o nosso projeto, que inclui enviar dados via MQTT para a aplicação web desenvolvida com Flask. Nesta aplicação, possui o recebimento e o envio de dados ao servidor.

O código desenvolvido está presente no site do wokwi:

<https://wokwi.com/projects/396226830608277505>

Durante o desenvolvimento, a maior diferença é que ao invés de implementá-la juntamente do flask, foi que utilizou-se um site para se inscrever nos tópicos, o HiveMQ, assim como no trabalho, o código manteve o mesmo. Sendo os sensores, o ultrassônico e o DHT22 (Temperatura e Umidade) e os atuadores o Buzzer ou Piezo, juntamente de um led vermelho.

O funcionamento do código é simples, se as condições de temperatura e umidade forem atingidas, isto é as condições forem propícias para a proliferação de mosquitos da dengue, os atuadores entram em ação, ativando uma frequência sonora que deixa os insetos atordoados e a luz led é ligada, devido a ela não ser quente.

Isso fez com que não somente o projeto conseguisse ser testado sem que o PjBL2 estivesse concluído, como também auxiliou na integração do projeto, pois precisar fazer tudo de uma única vez se tornaria mais massante.

6.3 TDE3 E TDE4

Na atividade “TDE4 – flask login com role” foi desenvolvido uma aplicação que permitia o acesso de acordo com o usuário, utilizando o tutorial disponibilizado no site da DigitalOcean. O passo a passo do site, permitia que fosse usado a biblioteca do flask-login para cada sessão, construir senhas com hash, garantindo mais segurança no banco de dados e a criação da tabela usuário no banco de dados.

Ao iniciar foi preciso a instalação de diversos pacotes, como flask, flask-login e flask-SQLAlchemy. Além disso, é preciso adicionar a pasta auth e ativar ela, por meio desses comandos:

```
PS C:\Users\danie\OneDrive\Documentos\DANI\PUCPR\3periodo\experiencia_criativa\teste\flask_auth_app\flask_auth_app\flask_auth_app> cmd
Microsoft Windows [versão 10.0.22631.3593]
(c) Microsoft Corporation. Todos os direitos reservados.

(auth) C:\Users\danie\OneDrive\Documentos\DANI\PUCPR\3periodo\experiencia_criativa\teste\flask_auth_app\flask_auth_app\flask_auth_app>auth\Scripts\activate

(auth) C:\Users\danie\OneDrive\Documentos\DANI\PUCPR\3periodo\experiencia_criativa\teste\flask_auth_app\flask_auth_app\flask_auth_app>set FLASK_APP=project

(auth) C:\Users\danie\OneDrive\Documentos\DANI\PUCPR\3periodo\experiencia_criativa\teste\flask_auth_app\flask_auth_app\flask_auth_app>flask run
* Serving Flask app 'project'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server.
```

Figura 18 - Flask run

O tutorial fornece todos os códigos, basta apenas colocar no lugar corretamente e corrigir eventuais erros. Assim, ao final na aplicação a pasta de arquivos ficará deste jeito:

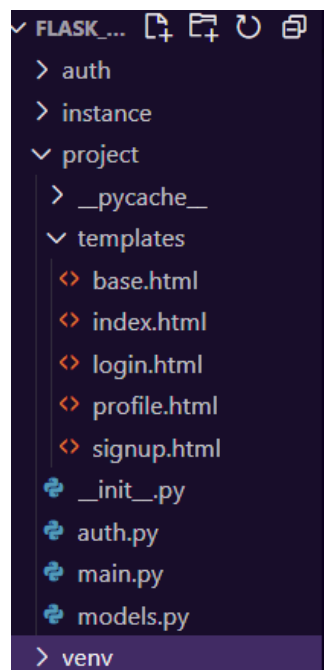


Figura 19 - Disposição do projeto Login

O usuário pode realizar seu cadastro na página de “Sign Up”, colocando o email, nome e senha:

Sign Up

user2@pucpr.edu.br

user2

...

Sign Up

Figura 20 - Sing up

Esses dados serão armazenados diretamente no banco de dados, guardando a senha criptografada, garantindo mais segurança ao usuário:

MySQL Workbench

Navigator

SCHEMAS

Filter objects

dbname

Tables

user

Columns

Indexes

Foreign Keys

Triggers

Views

Stored Procedures

Functions

estudo

games_store.database

gym

loja_katchau

Administration Schemas

Information

Table: user

Columns:

id int AI PK

email varchar(100)

password varchar(300)

name varchar(1000)

Query: SELECT * FROM dbname.user;

Result Grid

id	email	password	name
1	user1@gmail.com	crypt:32768:8:1\$m4PrSgh6ZR7mAFzI\$c1179d...	nome
2	user2@pucpr.edu.br	crypt:32768:8:1\$X1tY83nuya6rAk3N\$27c94dd...	user2

user 2 x

Apply Revert

Figura 21 - Banco de dados inserido

Então, ao obter sucesso na criação, o usuário pode realizar seu login com o email e a senha criada anteriormente por ele:

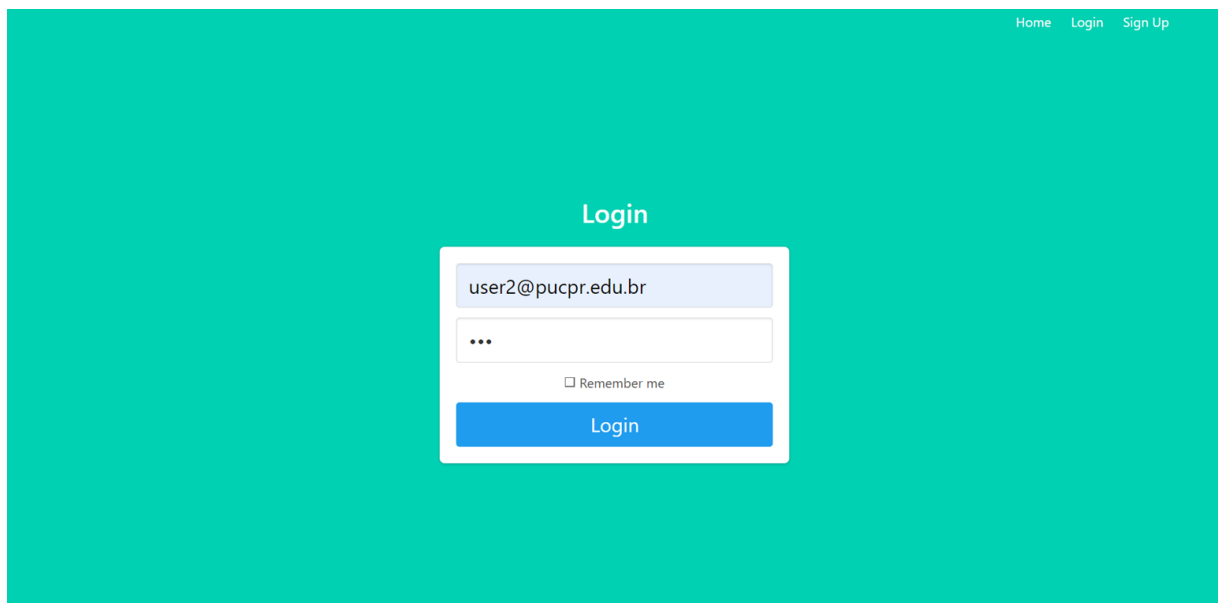


Figura 22 - Logando
Assim, ao se autenticar corretamente o usuário verá a página de perfil:

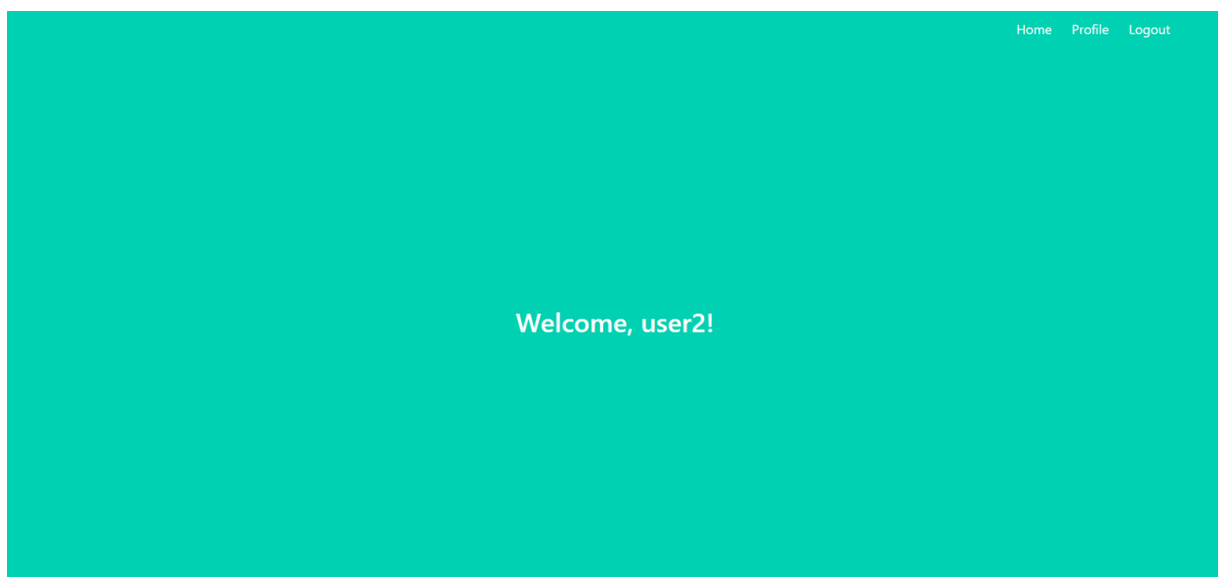


Figura 23 - Bem vindo!

7 CONCLUSÃO

O projeto “BuzzAway: o único buzz na sua casa” representa uma revolução no combate aos mosquitos, que são responsáveis pela transmissão de diversas doenças. Durante o seu desenvolvimento, o grupo conseguiu adquirir novas habilidades e conhecimentos, possibilitando a criação de uma nova ferramenta. O projeto tinha como objetivo desenvolver uma solução inteligente que combina hardware, aplicações web e banco de dados para mitigar os problemas causados por esses insetos.

Em hardware, o grupo foi capaz de integrar sensores de detecção de movimento, temperatura e umidade, essenciais para identificar a presença de mosquitos no ambiente. Também foi possível ter a experiência com atuadores, como o led e buzz.

No desenvolvimento de aplicações web, foi possível evoluir nossas habilidades já adquirida em semestres anteriores, tanto no front-end quanto no back-end, com o flask.

Ademais, com a integração com o banco de dados, com a parte web e hardware, é possível perceber os dados reunidos e assim, interpretar padrões que os mosquitos aparecem e a tomar decisões mais assertivas na prevenção de surtos.

BuzzAway é o resultado de todos esses conhecimentos. Trata-se de uma solução inovadora projetada para resolver os problemas causados por mosquitos de maneira inteligente. A tecnologia integra hardware, software e banco dados para criar um sistema que detecta a presença de mosquitos. O sistema ao detectar a presença destes insetos no ambiente, avisa o usuário e ativa o atuador, para que assim, espante esses seres indesejados. A solução é agradável para o usuário e para o meio ambiente, pois não utiliza venenos e repelentes, além de possui baixa manutenção. Além disso, a aplicação web permite monitorar e controlar o sistema de qualquer lugar, oferecendo atualizações em tempo real e relatórios por data.

O projeto BuzzAway oferece uma solução completa e inteligente para os problemas causados por mosquitos, contribuindo para um ambiente mais seguro e saudável para seus usuários. A equipe adquiriu novas habilidades e conhecimentos, que permitiu a criação de um produto que resolve um problema grave existente dentro

da sociedade. Sendo assim, os criadores da empresa esperam que o produto traga uma melhor qualidade de vida para as pessoas.

8 REFERÊNCIAS

HERBERT, Anthony, **How to add authentication to your app with flask-login**, disponível em: <https://www.digitalocean.com/community/tutorials/how-to-add-authentication-to-your-app-with-flask-login>. Acessado em: 24 de maio de 2024.

COMPONENTS101. **DHT22 Sensor - Pinout, Specifications, Datasheet**. Disponível em: <https://components101.com/sensors/dht22-pinout-specs-datasheet>. Acesso em: 6 jun. 2024.

STARLIGHTSENSORS. **DHT22 Sensor: A Comprehensive Guide to Temperature and Humidity Monitoring**. Disponível em: <https://www.starlightsensors.com/dht22-sensor-a-comprehensive-guide-to-temperature-and-humidity-monitoring/>. Acesso em: 6 jun. 2024.

MQTT. **MQTT Specification**. Disponível em: <https://mqtt.org/mqtt-specification/>. Acesso em: 6 jun. 2024.

THINGSBOARD. **MQTT Device API Reference | ThingsBoard Community Edition**. Disponível em: <https://thingsboard.io/docs/reference/mqtt-api/>. Acesso em: 6 jun. 2024.

ESP32IO. **ESP32 Piezo Buzzer Tutorial**. Disponível em: https://esp32io.com/tutorials/esp32-piezo-buzzer#google_vignette. Acesso em: 6 jun. 2024.