SISTEMA DE GERENCIAMENTO DE LIXEIRAS UTILIZANDO ESP32

Lucas Pereira da Silva – Fundação Hermínio Ometto – lpsilva@fho.alunos.edu.br
Mauricio Acconcia Dias – Fundação Hermínio Ometto – macdias@fho.edu.br

Resumo

Atualmente, a tecnologia está onipresente, com o objetivo de auxiliar, otimizar e facilitar nosso cotidiano, trazendo inúmeros benefícios e contribuindo para a praticidade na vida humana. No entanto, o elevado custo de certas tecnologias impede que seu uso seja amplamente prático e explorado, como, por exemplo, no caso de lixeiras inteligentes. Este projeto tem como objetivo utilizar o microcontrolador ESP32, em conjunto com uma aplicação web, para realizar a gestão de lixeiras em empresas, condomínios, organizações e residências. O sistema será capaz de medir o volume das lixeiras e informar ao usuário o momento adequado para a retirada do lixo, contribuindo para a sustentabilidade e promovendo a consciência ambiental. O ESP32 possibilita a conexão à internet via Wi-Fi para a troca de dados entre o sensor e a aplicação web, além de ser um microcontrolador de baixo consumo de energia e alta flexibilidade. A aplicação web proporcionará autonomia ao usuário, oferecendo a possibilidade de verificar os dados de qualquer localidade, bastando acessar um navegador.

Palavras-Chaves: Tecnologia, ESP32, Aplicação Web.

1. Introdução

Ao longo das últimas décadas, o cenário tecnológico vivencia uma transformação constante impulsionada pela acelerada evolução das inovações e descobertas científicas. Este rápido progresso tem sido um motor para a contínua redefinição da sociedade, com impactos profundos e abrangentes em quase todos os aspectos da vida humana. No epicentro dessa revolução tecnológica está a incessante busca pelo desenvolvimento de produtos mais eficientes, versáteis e interconectados, capazes de atender às crescentes demandas de uma sociedade cada vez mais ávida por soluções inovadoras.

A tecnologia da informação está presente em uma vasta gama de ambientes, proporcionando não apenas oportunidades de melhoria, mas também benefícios substanciais nos processos e na otimização do tempo. Um exemplo claro disso são os sistemas integrados de gestão, que operam tanto em nível estratégico quanto operacional.

Muito se discute sobre o uso da tecnologia na atualidade, entretanto, em meio a tantos recursos que tal meio pode nos oferecer, observamos a falta de exploração para utilizá-lo com ainda mais ênfase, pois com esse recurso, otimizamos diversos setores e de forma inimaginável.

Compreendemos que a tecnologia desempenha um papel significativo em diversos setores e nichos, tanto profissionais quanto pessoais. No entanto, ela ainda é considerada um recurso custoso, devido ao seu elevado nível de complexidade e à necessidade de especialização. Diante disso, o objetivo do presente trabalho é propor uma ferramenta útil e viável, que possa ser acessível aos consumidores de forma ampla, sem distinção de classes sociais. Assim, o desenvolvimento será orientado pelas necessidades dos usuários e com um excelente custo-benefício.

Considerando a otimização do tempo e a administração de um aspecto frequentemente omitido, visto que, não se fala em melhorias que irá contemplar tal nicho, o presente projeto propõe uma inovação para o gerenciamento de lixeiras empresariais (conforme a necessidade). Por meio do microcontrolador ESP32, em conjunto com um software web, será implementado um sistema para controlar principalmente o volume dos resíduos e facilitar o gerenciamento pelos usuários.

O projeto com o microcontrolador ESP32 permite a conexão à internet via WiFi, proporcionando uma maior praticidade na utilização do sistema. Dada a crescente acessibilidade da internet, especialmente em organizações onde seu uso é indispensável, esta solução se mostra altamente viável e eficiente.

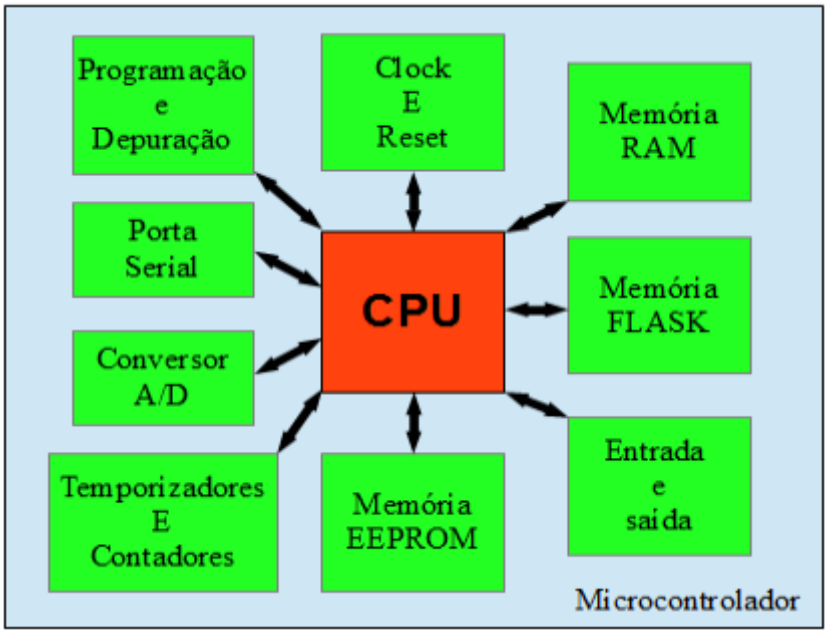
2. Referência bibliográfica

2.1 Microcontroladores

Segundo Kerschbaumer (2013), os microcontroladores são computadores de um único chip. Diferentemente dos computadores convencionais, que contêm periféricos para garantir seu funcionamento eficiente, os microcontroladores já possuem em sua arquitetura a CPU, a memória de dados, o clock e a memória de programa. Devido a essa arquitetura especial, os microcontroladores tornam-se muito atrativos aos consumidores pela facilidade de utilização e pelo menor custo no mercado.

A figura 1 ilustra os componentes de um microcontrolador:

Figura 1 - Componentes de um microcontrolador



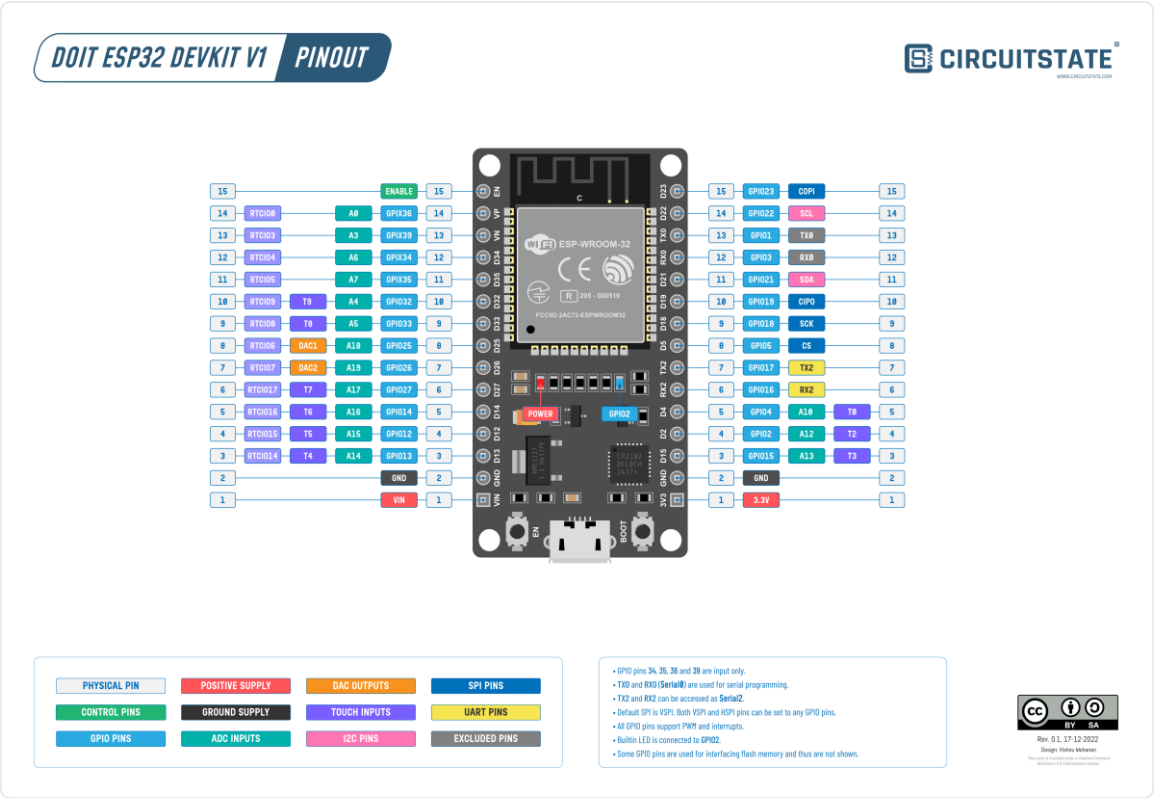
Fonte: Kerschbaumer (2013)

O microcontrolador é uma ferramenta altamente versátil, cuja aplicação pode ser encontrada em diversos setores, incluindo periféricos de computadores (como teclados, monitores e discos rígidos), eletrodomésticos (como televisores, máquinas de lavar e micro-ondas) e ambientes industriais (como balanças, controladores e sensores). Esta versatilidade deve-se à sua flexibilidade e baixo custo, proporcionando inúmeras possibilidades de uso, dependendo do software nele programado.

2.2 ESP32

O ESP32 é uma placa de desenvolvimento que integra uma série de microcontroladores, sendo escolhida para este projeto devido às suas características robustas. Esta placa oferece funcionalidades de Wi-Fi e Bluetooth, sendo o Wi-Fi de extrema importância para a conexão à internet e o envio de dados ao servidor. Além disso, o ESP32 se destaca pelo seu baixo consumo de energia e custo reduzido, tornando-o uma escolha ideal para vários tipos de aplicação.

Figura 2 – Diagrama de pinagem ESP32



Fonte: https://www.circuitstate.com/pinouts/doit-esp32-devkit-v1-wifi-development-board-pinout-diagram-and-reference/

2.3 Aplicação web

Uma aplicação web é definida como um software que é executado através de um navegador, eliminando a necessidade de instalação em um computador para acessar suas funcionalidades. Este tipo de aplicação apresenta diversos benefícios. Primeiramente, destaca-se a acessibilidade, permitindo que a aplicação seja facilmente acessada por qualquer dispositivo conectado à internet, independentemente da localização geográfica do usuário.

Além disso, a simplicidade para o usuário é um fator crucial, uma vez que não há necessidade de obter um equipamento específico com uma arquitetura determinada para seu funcionamento. O usuário também não precisa se preocupar com atualizações e questões de segurança do software, pois estas são gerenciadas de forma centralizada.

Outro aspecto importante é a escalabilidade, que possibilita o uso do sistema por um número ilimitado de usuários sem que a empresa precise se preocupar com infraestrutura e hardware.

Isso é especialmente relevante, dado que a maioria das aplicações web está hospedada em nuvem, facilitando a expansão conforme a demanda cresce.

O funcionamento de uma aplicação web baseia-se no modelo cliente-servidor, uma arquitetura essencial para garantir a eficácia da aplicação. A arquitetura do lado cliente é centrada no usuário, permitindo interações como clicar em botões, visualizar imagens e vídeos, entre outras atividades. Para isso, o script do lado cliente deve ser adequadamente programado para proporcionar essas funcionalidades.

Por outro lado, a arquitetura do lado servidor é responsável por processar as requisições provenientes do lado cliente. Por exemplo, quando um cliente cadastra um novo produto clicando em um botão, o servidor recebe essa requisição e armazena os dados em um banco de dados.

Essa divisão clara de responsabilidades entre o cliente e o servidor é a base essencial para o funcionamento eficiente e eficaz de uma aplicação web.

2.4 Sistemas embarcados

Segundo Cunha (2007), um sistema embarcado consiste na integração de capacidade computacional dentro de um circuito integrado, equipamento ou sistema. Um sistema embarcado não se limita a ser apenas um computador; trata-se de uma solução mais robusta, projetada para executar uma tarefa específica. O usuário não tem acesso ao código programado internamente, mas interage com o equipamento por meio de teclado, display, aplicações e outros dispositivos de interface.

O sistema embarcado, por sua vez, opera de maneira distinta de um computador convencional, uma vez que sua finalidade é exercer uma única função, tornando-o fixo para apenas uma ação específica. Essa característica o classifica como um dispositivo sem flexibilidade.

Para que um sistema embarcado seja eficaz, ele deve apresentar características específicas. O peso e o tamanho devem ser minimizados, visando a redução de custos e a economia de espaço. Além disso, é crucial que o consumo de energia seja baixo, tornando-o mais atrativo em comparação com outros sistemas. A robustez do equipamento também é essencial, uma

vez que ele pode operar em ambientes adversos, sujeitos a umidade, calor, vibrações, radiação e outros fatores.

3. Metodologia

Neste capítulo será comentado a metodologia utilizada para a realização do projeto. O sistema de gestão de lixeira seguirá uma metodologia que envolve a utilização do ESP32 para coleta de dados sobre o volume de resíduos em uma lixeira residencial, escolhida como unidade de estudo, e a aplicação web que mostrará as informações coletadas de forma simples.

3.1. Requisitos do sistema

Para fornecer dados em tempo real ao usuário, o sistema foi desenvolvido de forma a facilitar a integração entre hardware e software. Assim, o ESP32 coleta os dados do sensor ultrassônico, referentes à distância do lixo em relação ao ponto de medição, converte esses dados em porcentagem de capacidade preenchida e os envia em formato JSON para a aplicação web. A aplicação consome esse JSON em um endpoint dedicado, registrando o volume da lixeira no momento da coleta e armazenando-o em um banco de dados ao qual está conectada.

3.2. Tecnologias e componentes utilizados

Neste projeto foram utilizados as seguintes tecnologias e componentes:

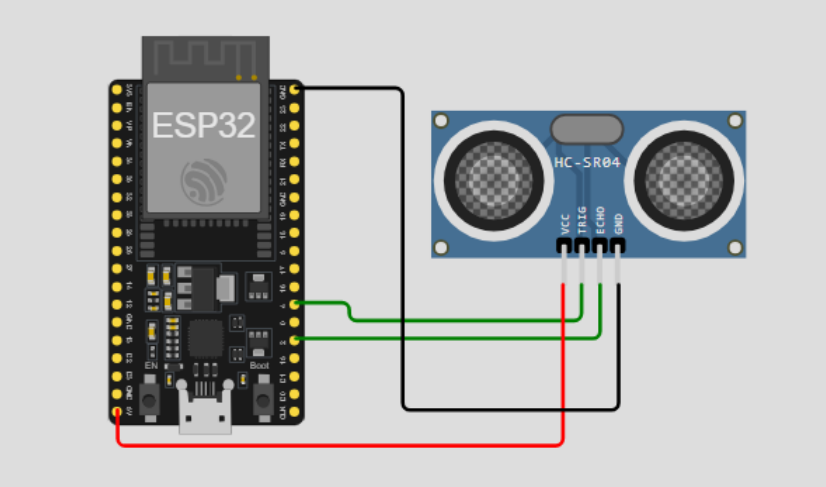
3.2.1. ESP32

O microcontrolador utilizado é a principal unidade do sistema, sendo responsável pela coleta e pelo envio dos dados ao servidor da aplicação web. Sua programação foi implementada por meio da plataforma de desenvolvimento Arduino IDE (versão 2.3.3).

3.2.2. Sensor HC-SR04

Para que o ESP32 possa coletar os dados, foi utilizado o sensor ultrassônico HC-SR04. Este sensor mede a distância entre os resíduos no interior da lixeira e a posição do sensor, utilizando para isso um pulso ultrassônico e aguardando o retorno do eco. A distância é calculada com base no tempo que o som leva para retornar ao sensor. O HC-SR04 é capaz de medir distâncias entre 2 cm e 4 m, o que o torna adequado para a aplicação em nosso projeto.

Figura 3 – Ligação do ESP32 e o sensor HC-SR04



Fonte: Elaborada pelo Autor (2024)

3.2.3. Arduino IDE

O Arduino IDE é uma ferramenta simples e eficaz para o desenvolvimento de códigos e compilações em placas de desenvolvimento, como Arduino Uno, ESP32, entre outras. Para o nosso projeto, estamos utilizando a versão 2.3.3 desta ferramenta.

3.2.4. Python

A linguagem de programação escolhida para o back-end foi Python, amplamente utilizada em diversas disciplinas do curso. Python será responsável por receber os dados coletados pelo sensor, armazená-los no banco de dados e disponibilizá-los para o front-end exibir ao usuário. A versão utilizada neste projeto é a 3.12.6.

3.2.5. Flask

Flask é um micro-framework escrito em Python, disponibilizado sob licença de código aberto, que permite a criação de APIs de maneira simples e rápida para nossa aplicação. Com o Flask, será possível criar rotas específicas para cada caso na nossa aplicação. A versão utilizada neste projeto é a 3.0.3.

3.2.6. SQLAlchemy

Para o armazenamento dos dados da nossa aplicação, juntamente com as informações coletadas pelos componentes eletrônicos, foi escolhido o SQLAlchemy. O SQLAlchemy é uma biblioteca de código aberto que implementa o mapeamento objeto-relacional (ORM). O ORM é uma técnica que facilita o trabalho dos desenvolvedores ao padronizar a correspondência entre classes e tabelas, e entre linhas de tabelas e instâncias de objetos. Com o SQLAlchemy, o desenvolvedor não precisa se preocupar diretamente com consultas,

inserções, atualizações e exclusões em SQL, pois todo o acesso ao banco de dados é realizado utilizando Python. A versão utilizada é a 2.0.35.

3.2.7. HTML

O HTML é uma linguagem de marcação mundialmente utilizada por aplicações web e que será o esqueleto da nossa aplicação.

3.2.8. Javascript

JavaScript é uma linguagem de programação avançada e uma das três tecnologias mais utilizadas no desenvolvimento web. Com ela, é possível dar vida a aplicações web por meio de interações na tela, atualizações dinâmicas, entre outros. Em nosso projeto, utilizaremos JavaScript especificamente para consumir dados do back-end, criar gráficos animados e atualizar dados automaticamente.

3.2.9. Bootstrap

Bootstrap é um framework front-end que facilita o desenvolvimento ao oferecer estruturas prontas de CSS e JavaScript para a criação de sites e aplicações. O que destaca o framework no cenário tecnológico é sua facilidade na implementação de recursos de responsividade, proporcionando uma melhor experiência visual ao usuário. Em nosso projeto, o Bootstrap será fundamental para a estética da aplicação, utilizando sua vasta gama de estilos para menus, tabelas, botões, fontes e cartões, bem como sua característica de responsividade. Além disso, utilizaremos suas funcionalidades em JavaScript para interações específicas de botões. A versão utilizada será a 5.3.

3.2.10. Chart.js

Considerado com uma biblioteca leve e flexível, o Chart.js tem por objetivo entregar gráficos responsivos e interativos para análises estatísticas, tendencias ou qualquer outro tipo de informação. O Chart.js oferece várias opções de gráficos como barra, linha, ponto, pizza, entre outros. A ferramenta se torna efetiva para nosso projeto, tendo como objetivo demonstrar dados obtidos através do sensor como um acompanhamento do volume e o volume atual. A versão utilizada será a 4.4.6.

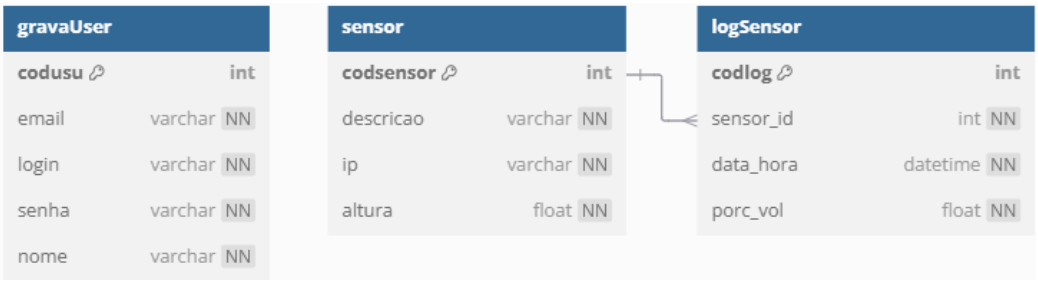
3.2.11. Visual Studio Code

Para o desenvolvimento de todo o código do back-end e do front-end, optamos pela IDE Visual Studio Code, uma ferramenta abrangente que atende a todos os requisitos do projeto e oferece suporte para as tecnologias utilizadas. A versão selecionada para uso será a 1.94.2.

3.3. Diagrama ERD

Na aplicação web, serão implementadas três tabelas principais: a tabela “grava_user”, que será responsável por armazenar os usuários cadastrados no sistema; a tabela “sensor”, encarregada de armazenar os sensores registrados; e a tabela “log_sensor”, que armazenará os volumes das lixeiras, bem como a data e o horário da coleta dos dados. Entre as tabelas “sensor” e “log_sensor”, haverá uma chave estrangeira que assegurará a relação entre os logs e os sensores correspondentes. Abaixo temos o diagrama ERD de nossa aplicação:

Figura 4 – Digrama ERD



Fonte: Elaborada pelo autor (2024)

4. Desenvolvimento

Neste capítulo iremos relatar toda a parte de desenvolvimento do sistema, partindo do hardware até o software.

4.1. Montagem do circuito

A montagem do circuito para o nosso sistema utilizou os seguintes componentes:

- ESP32;
- Sensor ultrassônico HC-SR04;
- Jumper
- Cabo Micro USB;
- Fonte USB 5V;

A conexão do ESP32 com o sensor ultrassônico foi realizada por meio de quatro jumpers fêmea-fêmea e quatro macho-fêmea, onde o pino VCC do sensor foi conectado ao pino VIN do microcontrolador, e o pino GND do sensor foi ligado ao pino GND do microcontrolador. Os pinos D2 e D4 da placa foram conectados aos pinos TRIG e ECHO do sensor,

respectivamente. Com essa conexão estabelecida corretamente, podemos utilizar o cabo Micro USB e a fonte de 5V para fornecer energia ao sistema.

4.2. Configuração wifi

Uma das funcionalidades mais importantes do ESP32 para garantir a eficiência do nosso projeto é a sua capacidade de conexão WiFi, que facilita a comunicação entre o servidor da aplicação e o microcontrolador. Para permitir essa conexão, configuramos o SSID e a senha da rede à qual o ESP32 deve se conectar e importamos a biblioteca WiFi.h para gerenciar a conexão. Esse processo de conexão está implementado no método setup(), com mensagens de status sendo exibidas no monitor serial para validação da conexão e apresentação do endereço IP obtido na rede conectada.

Figura 5 – Conexão WiFi (ESP32)

```
//Início conexão com o WIFI
Serial.println("Conectando a...");
Serial.println(ssid);

WiFi.begin(ssid,senha);
// Verifica conexão
while (WiFi.status() != WL_CONNECTED){
  delay(500);
  Serial.println(".");
}
Serial.println("");
Serial.println("WiFi Conectado");
Serial.println("IP: ");
Serial.println(WiFi.localIP());
```

Fonte: Elaborado pelo Autor (2024)

4.3. Configuração sensor ultrassônico

Para realizar a leitura dos dados coletados pelo sensor HC-SR04, é necessário, primeiramente, definir os pinos `trig` e `echo` do sensor, conforme ilustrado na figura abaixo:

Figura 6 – Definição de pinagem (HC-SR04)

```
const int PINO_TRIG = 4; // Pino D4 conectado ao TRIG do HC-SR04
const int PINO_ECHO = 2; // Pino D2 conectado ao ECHO do HC-SR04
```

Fonte: Elaborada pelo Autor (2024)

O pino trig atua como o sinal de saída do sensor, enquanto o pino echo é o sinal de entrada. Com essa configuração, é possível determinar a distância até um objeto utilizando a seguinte fórmula:

Distância = (Tempo de duração do sinal de saída * velocidade do som) / 2

Aplicando essa fórmula, podemos calcular a distância dos resíduos na lixeira até a posição do sensor. Para implementar essa funcionalidade no código, definimos inicialmente qual pino está associado a qual modo no método setup().

Figura 7 – Inicializa pinMode (HC-SR04)

```
pinMode(PINO_TRIG, OUTPUT); // Configura o pino TRIG como saída
pinMode(PINO_ECHO, INPUT); // Configura o pino ECHO como entrada
```

Fonte: Elaborada pelo Autor (2024)

Dessa forma, podemos detectar quando o pino trig está operando no modo alto utilizando a função digitalWrite e medir a duração do sinal com a função pulseIn ao receber o sinal no pino echo. Com essas informações, podemos calcular e apresentar a distância no monitor serial utilizando a fórmula mencionada anteriormente. Toda essa operação está implementada no método loop().

Figura 8 – Cálculo de Distância (HC-SR04)

```
digitalWrite(PINO_TRIG, LOW);
delayMicroseconds(10);
digitalWrite(PINO_TRIG, HIGH);
delayMicroseconds(10);
digitalWrite(PINO_TRIG, LOW);

long duracao = pulseIn(PINO_ECHO, HIGH);
float distancia = (duracao * 0.0343) / 2;
Serial.print("Distância: ");
Serial.print(distancia);
Serial.println(" cm");
```

Fonte: Elaborada pelo Autor (2024)

4.4. Envio de dados

Nosso sistema tem como objetivo apresentar ao usuário os dados referentes ao volume atual da lixeira no momento da consulta, além de armazenar um histórico com as datas e horários, juntamente com o volume registrado nesses momentos.

No código do ESP32, existe uma função denominada enviaLogSensor. Esta função verifica se o microcontrolador está conectado e, em seguida, realiza uma requisição utilizando a biblioteca HTTPClient.h. Através do método POST, a função envia um JSON contendo as informações do ID do sensor e a porcentagem do volume para o endpoint status, que está disponível no back-end da aplicação.

Figura 9 – Função para envio de dados a aplicação

```
void enviaLogSensor(int sensor_id, float porc_vol) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;

    http.begin(serverUrl);
    http.addHeader("Content-Type", "application/json");

    // Cria o JSON para enviar
    String jsonData = "{\"sensor_id\": " + String(sensor_id) + ", \"porc_vol\": " + String(porc_vol) + "}";

    // Envia o pedido POST
    int httpResponseCode = http.POST(jsonData);

    // Verifica a resposta do servidor
    if (httpResponseCode > 0) {
      String response = http.getString();
      Serial.println(httpResponseCode);
      Serial.println(response);
    } else {
      Serial.print("Erro ao enviar POST: ");
      Serial.println(httpResponseCode);
    }

    http.end();
  } else {
    Serial.println("Erro na conexão Wi-Fi");
  }
}
```

Fonte: Elaborada pelo Autor (2024)

A função enviaLogSensor é chamada no método loop(), onde ocorre a conversão da distância recebida pelo sensor em uma porcentagem, utilizando uma regra de três simples.

Figura 10 – Chamada da função

```
// Calcula capacidade
float capacidade = 100 - ((distancia * 100) / altura);
// Arredonda para duas casas
capacidade = round(capacidade * 100.0) / 100.0;
// Valida capacidade > 0
if (capacidade >= 0){
  enviaLogSensor(1, capacidade);
}
```

Fonte: Elaborada pelo Autor (2024)

4.5. Banco de dados

Utilizando o SQLAlchemy como banco de dados para nossa aplicação, conforme demonstrado no Diagrama de Entidade-Relacionamento (ERD) apresentado, ficou evidente a facilidade na criação de tabelas e seus campos no projeto. A funcionalidade ORM proporciona clareza na definição das tabelas por meio da criação de classes em Python, que representam essas tabelas. Portanto, em nossa programação, utilizando a biblioteca Flask-SQLAlchemy, iniciamos nosso servidor criando o banco de dados denominado "bin". Em seguida, definimos as classes com os respectivos campos das tabelas, levando em consideração todas as definições de banco de dados, como chave primária, autoincremento, chave estrangeira, tipo de dado, entre outros.

Figura 11 – Configuração Banco de Dados

```
#Configura Banco de dados
servidor.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///bin.db"

orm = SQLAlchemy()
#Cria tabelas
You, 2 weeks ago | 1 author (You)
class gravaUser(orm.Model) :
    codusu = orm.Column(orm.Integer, primary_key = True, autoincrement = True)
    email = orm.Column(orm.String, nullable = False,unique=True)
    login = orm.Column(orm.String, nullable = False)
    senha = orm.Column(orm.String, nullable = False)
    nome = orm.Column(orm.String, nullable = False)

You, 2 weeks ago | 1 author (You)
class sensor(orm.Model) :
    codsensor = orm.Column(orm.Integer, primary_key = True, autoincrement = True)
    descricao = orm.Column(orm.String, nullable = False)
    ip = orm.Column(orm.String, nullable = False)
    altura = orm.Column(orm.Float, nullable = False)
    logs = orm.relationship('logSensor', backref='sensor', lazy=True)

You, 3 weeks ago | 1 author (You)
class logSensor(orm.Model):
    codlog = orm.Column(orm.Integer, primary_key=True, autoincrement=True)
    sensor_id = orm.Column(orm.Integer, orm.ForeignKey('sensor.codsensor'), nullable=False)
    data_hora = orm.Column(orm.DateTime, default=lambda: datetime.now(brt),nullable=False)
    porc_vol = orm.Column(orm.Float, nullable=False)

orm.init_app(servidor)
```

Fonte: Elaborada pelo Autor (2024)

4.6. Rotas API

Em nossa aplicação, desenvolvida utilizando o framework Flask, implementamos uma API que contém diversas rotas, cada uma com um propósito específico dentro da aplicação. Essas rotas podem realizar desde consultas simples de dados até operações mais complexas, como o cadastro de usuários. Cada rota foi projetada como uma função em Python, associada a um método HTTP específico: GET para consultas em tabelas, POST para inserção de dados, PUT para atualização de dados, e DELETE para exclusão de registros.

Dentro das funções, garantimos que as informações recebidas no corpo da requisição sejam obrigatoriamente em formato JSON. Posteriormente, tratamos essas informações para realizar a operação solicitada. As respostas das rotas são estruturadas em JSON, podendo incluir códigos de status, mensagens de sucesso e outros dados relevantes. Além disso, implementamos validações de exceções para assegurar o retorno de códigos de erro padrão conforme necessário.

As operações no banco de dados são executadas dentro da função correspondente. Em casos de requisições POST, por exemplo, uma sessão no banco de dados é aberta para realizar a operação de inserção e, logo após, é finalizada. Caso ocorram erros durante o processo, a sessão é revertida por meio de um rollback.

Os códigos de resposta da API se classificam conforme a lista abaixo:

- 200 – OK - Solicitação bem-sucedida;
- 201 – Created - Solicitação bem-sucedida e um novo recurso foi criado;
- 400 – Bad Request – Não foi possível interpretar a requisição;
- 401 – Unathourized – A requisição requer a autenticação do usuário;
- 500 – Internal Server Error – Ocorreu um erro no servidor;

A resposta fornecida ao cliente será determinada pela rota solicitada, podendo consistir em uma mensagem de sucesso, um código interno ou a própria exceção capturada pelo tratamento de erros.

Figura 12 – Rota Cadastro de Usuário

```python
#cadastro de usuários
@servidor.route("/cadastrar",methods =["POST"])
def cadastrar_usuario():
    dados = request.get_json()

    # Hash da senha antes de armazenar
    hashed_password = bcrypt.hashpw(dados['senha'].encode('utf-8'), bcrypt.gensalt())

    user = gravaUser(
        email=dados['email'],
        login = dados['login'],
        senha=hashed_password.decode('utf-8'),
        nome=dados['nome']
    )

    try:
        orm.session.add(user)
        orm.session.commit()
        orm.session.refresh(user)
        response = {"codigo_usuario": user.codusu}, 201
    except Exception as e:
        orm.session.rollback()  # Reverte qualquer mudança no banco de dados
        response = jsonify({"mensagem": "Erro ao cadastrar usuário", "erro": str(e)}), 500

    return response
```

Fonte: Elaborada pelo Autor (2024)

4.6. Dashboard

Com o sensor configurado, as informações gravadas no banco de dados e as rotas criadas conforme o método desejado, a visualização de todo esse processo de maneira intuitiva e analítica precisava ser entregue pela nossa aplicação. Assim, foram desenvolvidos gráficos e cards com o objetivo de apresentar ao cliente informações básicas, como volume da lixeira, quantidade de usuários cadastrados, e entre outras.

Para que o front-end pudesse obter essas informações, foi utilizado JavaScript, uma linguagem de programação essencial para o desenvolvimento web. As requisições à API desenvolvida foram incorporadas em funções assíncronas, permitindo ao cliente utilizar o sistema sem precisar aguardar a conclusão de uma requisição, conferindo a autonomia necessária para que os dados sejam exibidos de forma recíproca entre servidor e cliente.

Na construção dos gráficos, utilizamos o Chart.js, proporcionando a criação de gráficos intuitivos e em tempo real. Os dados foram obtidos através da API, permitindo a definição do tipo de gráfico e dos pontos de amostra de maneira dinâmica e precisa.

Figura 13 – Função Gráfico (Chart.js)

```
async function createCharts(maxSensor) {
  // Configuração do gráfico de donut
  const xValues = ["Volume Livre", "Volume Ocupado"];
  const yValues = [100 - maxSensor, maxSensor];
  const barColors = ["#00aba9", "#dc3545"];

  new Chart('myChart', {
    type: 'doughnut',
    data: {
      labels: xValues,
      datasets: [{
        backgroundColor: barColors,
        data: yValues
      }]
    }
  });
```

Fonte: Elaborada pelo Autor (2024)

4.6. CRUD

CRUD é um acrônimo para as operações básicas de armazenamento de dados: Create (Criar), Read (Ler), Update (Atualizar) e Delete (Excluir). Em nossa aplicação, o CRUD foi amplamente implementado, disponibilizando funções essenciais que todo sistema deve conter. O front-end realiza requisições, enviando o comando a ser executado e retornando o resultado para o usuário.

Figura 14 – Função de Exclusão de usuário

```
async function excluirUsuario(event) {
    event.preventDefault();
    const form = document.getElementById("cadastroForm2");
    const formData = new FormData(form);
    const codusu = formData.get("codusu");

    const response = await fetch("/user/" + codusu, {
        method: "DELETE",
        headers: {
            "Content-Type": "application/json",
        },
    });

    const result = await response.json();
    if (response.ok) {
        alert("Usuario excluído com sucesso!")
        window.location.href = "/user";
    } else {
        alert("Erro: " + result.mensagem);
    }
}
```

Fonte: Elaborada pelo Autor (2024)

5. Resultados

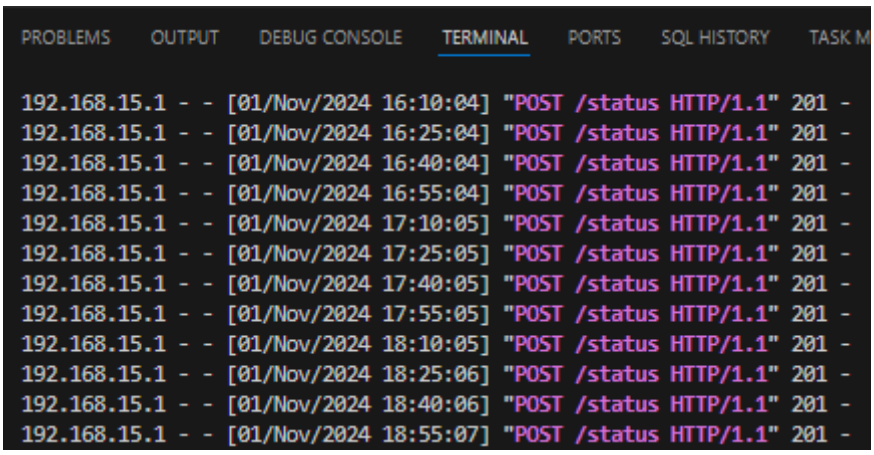
Na parte de hardware, o sistema operou de maneira satisfatória, estabelecendo a conexão Wi-Fi de forma rápida e coletando os dados de distância conforme o esperado. A capacidade do dispositivo de se comunicar com o servidor da API foi validada, apresentando erros apenas quando o servidor estava indisponível. No entanto, foram observadas variações repentinas nos valores capturados pelo sensor em alguns casos, onde um valor inicialmente alto diminuiu repentinamente. Isso provavelmente ocorreu devido ao fato de a lixeira não estar fixada, tornando-se suscetível a deslocamentos momentâneos.

Figura 15 – Tabela Log do Sensor

codlog	sensor_id	data_hora	porc_vol
151	1	2024-10-31 14:04:52.334841	87.81
152	1	2024-10-31 14:19:52.435953	87.81
153	1	2024-10-31 14:34:52.849144	87.81
154	1	2024-10-31 14:49:52.949875	87.81
155	1	2024-10-31 15:04:53.048841	87.81
156	1	2024-10-31 15:19:53.155406	87.72
157	1	2024-10-31 15:34:53.564279	85.47
158	1	2024-10-31 15:49:53.668155	85.47
159	1	2024-10-31 16:04:53.768065	83.12

Fonte: Elaborada pelo Autor (2024)

Figura 16 – Retorno API



Fonte: Elaborada pelo Autor (2024)

Um dos objetivos do projeto foi a obtenção de um custo reduzido. Este resultado foi alcançado por meio da seleção cuidadosa dos componentes utilizados e da realização de pesquisas de preços ao longo do ano. O custo total pode ser analisado na Tabela 1.

Tabela 1 – Tabela de preço do projeto

Item	**Quantidade**	**Valor Unitário**
ESP32 Dev Kit 1	1	R$ 37,99
Sensor Ultrassônico HC-SR04	1	R$ 10,99
Jumper Fêmea / Fêmea	4	R$ 1,20
Jumper Macho / Fêmea	4	R$ 1,20
Cabo Micro USB	1	R$ 6,90
Fonte USB 5V	1	R$ 13,90
	TOTAL	R$ 72,18

Fonte: Elaborada pelo Autor (2024)

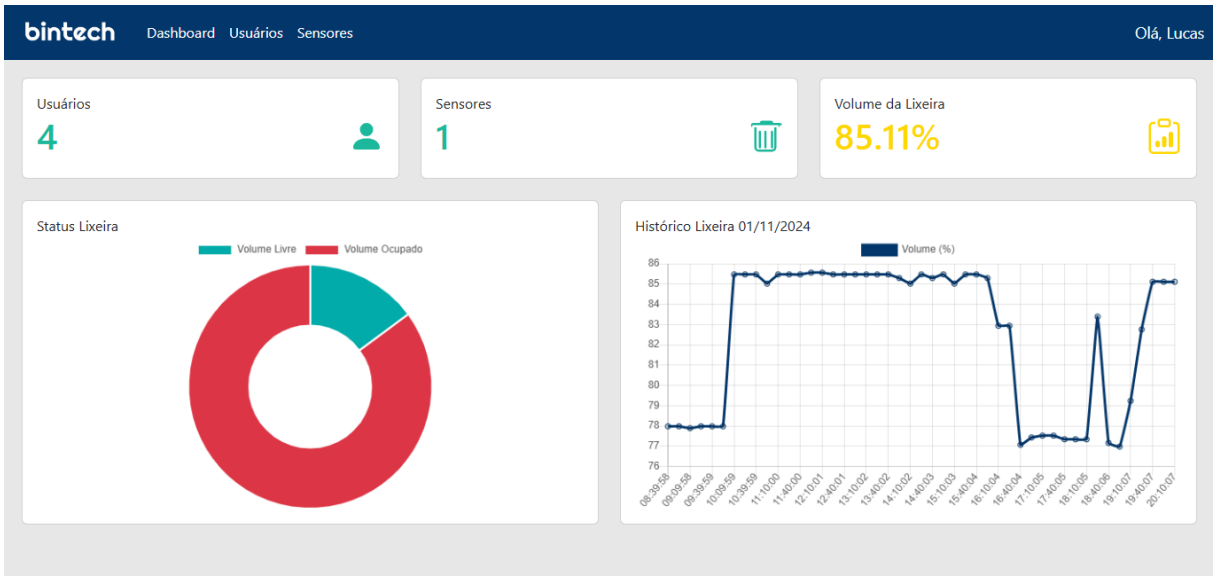
No software, o sistema operou de maneira altamente satisfatória em todos os aspectos, com a comunicação entre o servidor e o cliente sendo fundamental para uma apresentação eficaz. A utilização de frameworks como Bootstrap e Chart.js proporcionou uma interface visual limpa e fácil de usar. As requisições assíncronas fluíram perfeitamente, trazendo e enviando dados de maneira correta conforme as execuções solicitadas. A utilização do Flask como servidor de API, em conjunto com o SQLAlchemy, conferiu ao sistema rapidez na execução e facilidade de interpretação para o desenvolvedor.

Abaixo temos a apresentação de duas telas da nossa aplicação:

Figura 17 – Página de Login

Fonte: Elaborada pelo Autor (2024)

Figura 18 – Página Dashboard



Fonte: Elaborado pelo Autor (2024)

6. Conclusão

O projeto desenvolvido demonstra que tecnologias de alta performance e baixo custo podem operar de maneira eficiente em ambientes que tradicionalmente recebem poucos investimentos, como no caso das lixeiras. Apesar de algumas variações na coleta de dados em relação à distância dos resíduos, o sistema revela-se útil e aplicável em residências, empresas,

organizações e outros contextos. O principal benefício da proposta é a otimização do tempo na retirada de lixo, a qual pode ser facilmente visualizada por meio da aplicação web desenvolvida.

Um aspecto relevante é a valorização do projeto em relação aos dados, uma vez que todas as informações foram mapeadas de forma precisa, proporcionando uma experiência simples e dinâmica ao usuário, facilitando a compreensão do que é apresentado na tela. Ademais, o controle de custos é um ponto considerado positivamente pelos potenciais clientes do sistema.

Outra perspectiva a ser considerada é a sustentabilidade. Com a gestão adequada do sistema, é possível oferecer ao cliente uma avaliação precisa do volume de resíduos gerados dentro da empresa, promovendo a reciclagem e a reutilização de materiais, evitando assim o impacto negativo no meio ambiente e o desperdício de recursos.

Portanto, o sistema de gerenciamento utilizando o ESP32 se mostra extremamente útil para o cotidiano, contribuindo para um mundo mais sustentável, econômico e gerenciável, reafirmando o compromisso com a utilização da tecnologia em benefício da sociedade.

REFERÊNCIAS

ALBERTIN, Alberto Luiz; ALBERTIN, Rosa Maria de Moura. Benefícios do uso de tecnologia de informação para o desempenho empresarial. Revista de Administração Pública, v. 42, p. 275-302, 2008.

ISZCZUK, Ana Claudia Duarte et al. Evoluções das tecnologias da indústria 4.0: dificuldades e oportunidades para as micro e pequenas empresas. Brazilian Journal of Development, v. 7, n. 5, p. 50614-50637, 2021.

KERSCHBAUMER, Ricardo et al. Microcontroladores. Santa Catarina, Brasil, 2013.

O que é uma aplicação Web? – Explicação sobre aplicações Web — AWS. Disponível em: <https://aws.amazon.com/pt/what-is/web-application/>.

CUNHA, Alessandro F. O que são sistemas embarcados. Saber Eletrônica, v. 43, n. 414, p. 1-6, 2007.

CONTRIBUTORS, M. O., Jacob Thornton, and Bootstrap. Get started with Bootstrap. Disponível em: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>.

Welcome to Flask — Flask Documentation (3.0.x). Disponível em: <https://flask.palletsprojects.com/en/stable/>.

Chart.js documentation. Disponível em: <https://www.chartjs.org/docs/latest/>.

SQLAlchemy Documentation — SQLAlchemy 2.0 Documentation. Disponível em: <https://docs.sqlalchemy.org/en/20/>.

PYTHON SOFTWARE FOUNDATION. Welcome to Python.org. Disponível em: <https://www.python.org/doc/>.