**Documentação do Código**

**CSS (styles.css)**

O CSS é responsável pela estilização da página web. A seguir estão as classes e suas descrições:

**1. body**

css

Copiar código

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 20px;

background-color: #f0f0f0;

}

* **font-family**: Define a fonte padrão como Arial.
* **margin**: Remove margens externas da página.
* **padding**: Adiciona um espaçamento interno de 20 pixels.
* **background-color**: Define a cor de fundo da página como cinza claro.

**2. h1**

css

Copiar código

h1 {

color: #333;

}

* **color**: Define a cor do texto do cabeçalho h1 como um tom escuro de cinza.

**3. input**

css

Copiar código

input {

display: block;

margin: 10px 0;

}

* **display**: Define o elemento input como um bloco, fazendo com que ocupe toda a largura disponível.
* **margin**: Adiciona um espaçamento de 10 pixels acima e abaixo dos inputs.

**4. button**

css

Copiar código

button {

padding: 10px;

background-color: #007bff;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

* **padding**: Adiciona um espaçamento interno de 10 pixels.
* **background-color**: Define a cor de fundo do botão como azul.
* **color**: Define a cor do texto como branca.
* **border**: Remove a borda padrão do botão.
* **border-radius**: Arredonda as bordas do botão em 5 pixels.
* **cursor**: Muda o cursor para indicar que o botão é clicável.

**5. button**

css

Copiar código

button:hover {

background-color: #0056b3;

}

* **background-color**: Altera a cor de fundo do botão para um azul mais escuro quando o mouse está sobre ele.

**JavaScript (timer.js)**

O JavaScript é responsável pela lógica do timer e pela interação com o backend.

**1. updateTimer()**

javascript

Copiar código

function updateTimer() {

fetch('/api/time\_remaining')

.then(response => response.json())

.then(data => {

const timeElement = document.getElementById('time-remaining');

timeElement.innerText = data.time\_remaining;

if (data.time\_remaining <= 0) {

// Ações a serem executadas quando o timer chega a zero

}

})

.catch(error => console.error('Erro ao obter o tempo restante:', error));

}

* **fetch('/api/time\_remaining')**: Faz uma requisição para obter o tempo restante até o próximo relatório.
* **response.json()**: Converte a resposta em JSON.
* **timeElement.innerText**: Atualiza o texto do elemento com o id time-remaining com o tempo restante.
* **if (data.time\_remaining <= 0)**: Verifica se o tempo restante é menor ou igual a zero para executar ações específicas.

**2. setInterval(updateTimer, 1000)**

* Inicia um loop que chama a função updateTimer a cada 1000 milissegundos (1 segundo).

**3. stopScheduler()**

javascript

Copiar código

function stopScheduler() {

window.location.href = '/stop';

}

* Redireciona o usuário para o endpoint /stop, que para o agendador de tarefas.

**HTML (index.html e timer.html)**

**1. index.html**

html

Copiar código

<!DOCTYPE html>

<html lang="pt-BR">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

<title>Upload de CSV</title>

</head>

<body>

<h1>Upload de CSV</h1>

<form method="POST" enctype="multipart/form-data">

<input type="file" name="file" accept=".csv" required>

<input type="text" name="pdf\_name" placeholder="Nome do PDF" required>

<input type="text" name="pdf\_save\_path" placeholder="Caminho para salvar o PDF">

<select name="frequency" required>

<option value="minute">A cada minuto</option>

<option value="hour">A cada hora</option>

<option value="day">Diariamente</option>

<option value="month">Mensalmente</option>

<option value="year">Anualmente</option>

</select>

<button type="submit">Enviar</button>

</form>

</body>

</html>

* **Cabeçalho**: Define a estrutura básica do documento HTML, incluindo a meta charset e a inclusão do CSS.
* **Formulário**: Permite o upload de um arquivo CSV e a configuração de um nome e caminho para o PDF, além da frequência de geração.

**2. timer.html**

html

Copiar código

<!DOCTYPE html>

<html lang="pt-BR">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="{{ url\_for('static', filename='css/styles.css') }}">

<script src="{{ url\_for('static', filename='js/timer.js') }}" defer></script>

<title>Timer de Relatório</title>

</head>

<body>

<h1>Tempo Restante para o Próximo Relatório</h1>

<p>O próximo relatório será gerado em <span id="time-remaining">{{ time\_remaining }}</span> segundos.</p>

<button onclick="stopScheduler()">Parar e Voltar</button>

</body>

</html>

* **Cabeçalho**: Inclui o JavaScript do timer e a folha de estilo CSS.
* **Timer**: Exibe o tempo restante até o próximo relatório e oferece um botão para parar o agendador.

**Python (Flask App)**

A aplicação Flask é responsável pela lógica de backend, manipulação de arquivos e agendamento de tarefas.

**1. Inicialização e Configurações**

python

Copiar código

app = Flask(\_\_name\_\_)

DB\_NAME = "vendas.db"

scheduler = BackgroundScheduler()

pdf\_save\_path = ""

next\_run\_interval = None

* **Flask**: Cria uma instância da aplicação.
* **Database**: Define o nome do banco de dados SQLite.
* **Scheduler**: Inicializa o agendador em segundo plano.
* **Variáveis**: Define variáveis globais para armazenar o caminho de salvamento do PDF e o intervalo de execução.

**2. Funções Principais**

* **init\_db()**: Cria o banco de dados e a tabela vendas se não existir.
* **generate\_graphs(df)**: Gera gráficos a partir do DataFrame recebido, salvando os gráficos como imagens.
* **generate\_report(pdf\_name)**: Gera um relatório em PDF com base nas vendas e nos gráficos gerados.
* **upload\_file()**: Endpoint para fazer upload de arquivos CSV e agendar a geração de relatórios.
* **time\_remaining()**: Endpoint que retorna o tempo restante até o próximo relatório em formato JSON.
* **timer()**: Renderiza a página do timer.
* **stop()**: Para todos os jobs agendados e reinicializa o banco de dados.
* **get\_vendas()**: Endpoint que retorna os dados das vendas armazenadas no banco de dados.

**3. Execução da Aplicação**

python

Copiar código

if \_\_name\_\_ == "\_\_main\_\_":

init\_db()

scheduler.start()

app.run(debug=True)

* **init\_db()**: Inicializa o banco de dados quando a aplicação é executada.
* **scheduler.start()**: Inicia o agendador de tarefas.
* **app.run(debug=True)**: Executa a aplicação em modo de depuração.