Tutorial Flask

Sumário

O que é Flask?

Flask é um microframework de desenvolvimento web escrito em Python. É chamado de "microframework" porque é leve e minimalista, oferecendo apenas os componentes essenciais necessários para criar aplicações web, como roteamento de URLs, tratamento de solicitações HTTP e templates. Apesar de ser "micro", Flask é altamente extensível e permite que desenvolvedores adicionem diversas funcionalidades conforme necessário, como bancos de dados, autenticação, e outras.

Principais Características do Flask:

1. **Simplicidade e Flexibilidade**: Flask não impõe muitos padrões ou restrições, permitindo que os desenvolvedores escolham como estruturar suas aplicações e quais bibliotecas usar. É ideal para projetos que começam pequenos e crescem em complexidade com o tempo.
2. **Modularidade**: Flask permite a adição de extensões para incluir recursos adicionais, como integração com bancos de dados (usando SQLAlchemy), autenticação de usuários, validação de formulários, etc.
3. **Roteamento de URLs**: Flask facilita a definição de rotas (URLs) para diferentes partes da aplicação usando o decorador `@app.route`.
4. **Templates Jinja2**: Flask usa o mecanismo de templates Jinja2 para renderizar páginas HTML de maneira dinâmica. Você pode usar variáveis, loops, e condicionalidades dentro dos templates para criar páginas web dinâmicas.
5. **Suporte a Métodos HTTP**: Flask permite que os desenvolvedores definam como as rotas respondem a diferentes métodos HTTP (GET, POST, PUT, DELETE), facilitando a criação de APIs RESTful.
6. **Ambiente de Desenvolvimento Integrado**: Flask possui um servidor de desenvolvimento integrado que reinicia automaticamente sua aplicação quando alterações são feitas, além de um depurador interativo.

Exemplo de uma Aplicação Simples com Flask

```
from flask import Flask, render_template, request, redirect,
url_for

app = Flask(__name__)

@app.route('/')
def index():
    return "Bem-vindo ao Flask!"

@app.route('/ola/<nome>')
def ola(nome):
    return f"Olá, {nome}!"

if __name__ == '__main__':
```

```
app.run(debug=True)
```

Neste exemplo:
`Flask(__name__)` cria uma instância da aplicação Flask.
`@app.route('/')` define uma rota que responde à URL base (/).
A função `index()` retorna uma mensagem de boas-vindas.
`@app.route('/ola/<nome>')` define uma rota dinâmica que aceita um nome na URL e o exibe na página.

Implementando o exemplo acima

1. **Listar as versões instaladas do Python**:
   `py --list ou py -0`

2. **Criar o Diretório do Projeto**:
   Primeiro, crie um diretório para o seu projeto Flask:
   ```
   mkdir meu_app_flask
   cd meu_app_flask
   ```

3. **Criar um Ambiente Virtual**:
   É uma boa prática criar um ambiente virtual para isolar as dependências do projeto.
   Use o comando py para criar o ambiente virtual:
   `py -3.12 -m venv .venv`

   Ative o ambiente virtual:
   `.\venv\Scripts\activate`

4. **Instalar Flask**:
   Com o ambiente virtual ativado, instale o Flask:
   `pip install Flask`

5. **Criar o Arquivo app.py**:
   ```
   from flask import Flask, render_template, request, redirect,
   url_for

   app = Flask(__name__)

   @app.route('/')
   def index():
       return "Bem-vindo ao Flask!"

   @app.route('/ola/<nome>')
   def ola(nome):
       return f"Olá, {nome}!"
   ```

```
if __name__ == '__main__':
    app.run(debug=True)
```

6. **Adicionar Arquivos de Template (Opcional)**:
   Embora o código fornecido não use templates, você pode criar um diretório `templates` e adicionar arquivos HTML para renderizar páginas mais complexas no futuro. Crie um diretório `templates`:
   `mkdir templates`

7. **Executar o Servidor Flask**:
   Para iniciar a aplicação Flask, execute:
   `python app.py`

   Isso iniciará o servidor Flask em modo de desenvolvimento (`debug=True`), e você verá mensagens indicando que o servidor está rodando.

8. **Acessar a Aplicação**:
   Abra um navegador da web e acesse `http://127.0.0.1:5000/` para ver a mensagem "Bem-vindo ao Flask!".

   Acesse `http://127.0.0.1:5000/ola/SeuNome` substituindo **`SeuNome`** pelo nome que deseja exibir para ver a mensagem "Olá, SeuNome!".

Resumo da Estrutura de Pastas

A estrutura básica do seu projeto será:

```
meu_app_flask_app
│
├── app.py
├── venv
│   └── (diretório do ambiente virtual)
└── templates (opcional)
    └── (arquivos HTML para templates, se necessário)
```

Observações

- **Ambiente Virtual**: Manter o ambiente virtual ativado enquanto trabalha no projeto garante que as dependências instaladas (como Flask) não afetem outros projetos Python.
- **Debug Mode**: O Flask é executado no modo de depuração (debug=True) apenas para desenvolvimento. Para produção, você deve configurar o Flask de forma adequada e usar um servidor de produção, como o Gunicorn.

O que é Jinja2

Jinja2 é o mecanismo de templates utilizado pelo Flask para gerar HTML dinâmico e renderizar templates. Ele permite que você insira variáveis e lógica de controle em seus arquivos HTML, facilitando a criação de páginas web dinâmicas e interativas.

Aqui estão alguns pontos-chave sobre o Jinja2 no Flask:

1. **Mecanismo de Templates**
    - **O que é**: Jinja2 é uma biblioteca de templates para Python que permite separar a lógica da apresentação. É usado para criar arquivos HTML que podem conter placeholders para variáveis, estruturas de controle e loops.
    - **Como funciona**: Jinja2 processa os arquivos de template, substituindo placeholders e avaliando expressões para gerar HTML que é enviado para o navegador.

2. **Principais Recursos do Jinja2**
    - **Variáveis**: Você pode inserir valores dinâmicos em seus templates com o uso de variáveis. Exemplo:
      ```
      {{ usuario.nome }}
      ```

    - **Condicionais**: Permite exibir ou ocultar partes do template com base em condições. Exemplo:
      ```
      {% if esta_logado %}
          <p>Bem vindo de volta!</p>
      {% else %}
          <p>Faça login.</p>
      {% endif %}
      ```

    - **Loops**: Permite iterar sobre listas e outras estruturas de dados. Exemplo:
      ```
      <ul>
          {% for item in items %}
              <li>{{ item }}</li>
          {% endfor %}
      </ul>
      ```

    - **Filtros**: Modificam a forma como os dados são exibidos. Exemplo:
      ```
      <p>{{ "2024-07-25" | date("F j, Y") }}</p>
      ```

      Aqui, **`date`** é um filtro que formata a data.

    - **Blocos e Herança**: Permite criar um layout base e estender ou modificar partes dele em outros templates. Exemplo:
      ```
      {% extends 'base.html' %}
      {% block content %}
          <h1>Conteúdo da página</h1>
      {% endblock %}
      ```

3. **Como Jinja2 se Integra com Flask**
   - **Uso no Flask**: Quando você usa a função `render_template` no Flask, o Jinja2 é responsável por processar os templates e gerar o HTML final que será enviado ao cliente.
   - **Templates**: Os arquivos de template geralmente são armazenados na pasta `templates` do seu projeto Flask. O Flask automaticamente procura por templates nesta pasta.

4. **Exemplo Básico**

   Estrutura do Projeto

   Vamos criar a seguinte estrutura de pastas para a aplicação:

```
flask_jinja_exemplo
│
├── app.py
├── templates
│   ├── base.html
│   ├── index.html
│   └── detalhes.html
├── static/
│   └── style.css
└── requirements.txt
```

   1. **Código Flask (app.py)**

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    usuario = {'nome': 'Gustavo', 'idade': 4}
    items = ['Maçãs', 'Bananas', 'Cerejas']
    contexto = {
        'usuario': usuario,
        'items': items,
        'esta_logado': True,
        'dado': {'nome': 'Flask', 'versao': '2.0'}
    }
    return render_template('index.html', **contexto)

@app.route('/detalhes/')
def detalhes():
    usuario = {'nome': 'Godofredo', 'idade': 25}
    return render_template('detalhes.html', usuario=usuario)

if __name__ == '__main__':
    app.run(debug=True)
```

2. Templates

Base Template (`templates/base.html`)
O arquivo **`base.html`** será um template base que outros templates podem herdar:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>{% block title %}Meu aplicativo Flask{% endblock
%}</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">
</head>
<body>
    <header>
        <h1>Meu aplicativo Flask</h1>
        <nav>
            <a href="{{ url_for('index') }}">Home</a>
            <a href="{{ url_for('detalhes') }}">Detalhes</a>
        </nav>
    </header>
    <main>
        {% block content %}{% endblock %}
    </main>
</body>
</html>
```

Index Template (`templates/index.html`)
O arquivo **`index.html`** usa variáveis, condicionais, listas, dicionários e filtros:

```
{% extends 'base.html' %}

{% block title %}Home{% endblock %}

{% block content %}
    <h2>Bem-vindo, {{ usuario.nome | upper }}!</h2>
    <p>Idade: {{ usuario.idade }}</p>


    {% if esta_logado %}
        <p>Você está logado.</p>
    {% else %}
        <p>Faça login.</p>
    {% endif %}
```

```
    <h3>Items:</h3>
    <ul>
        {% for item in items %}
            <li>{{ item }}</li>
        {% endfor %}
    </ul>

    <h3>Dicionário de dados:</h3>
    <p>Nome: {{ dado.nome }}</p>
    <p>Versão: {{ dado.versao }}</p>
{% endblock %}
```

Detalhes Template (`templates/detalhes.html`)

O arquivo `detalhes.html` demonstra o uso de variáveis simples e herança de templates:

```
{% extends 'base.html' %}

{% block title %}Detalhes{% endblock %}

{% block content %}
    <h2>Detalhes do usuário</h2>
    <p>Nome: {{ usuario.nome }}</p>
    <p>Idade: {{ usuario.idade }}</p>
{% endblock %}
```

3. **Arquivo de Estilos (opcional)**

   Você pode adicionar um arquivo CSS para estilizar sua aplicação. Crie o arquivo `static/style.css`:

```
body {
    font-family: Arial, sans-serif;
}

header {
    background-color: #f8f9fa;
    padding: 10px;
    text-align: center;
}

nav a {
    margin: 0 10px;
    text-decoration: none;
    color: #007bff;
}

main {
    padding: 20px;
}
```

4. **Arquivo de Dependências**
   Liste as dependências necessárias em um arquivo `requirements.txt`:
   `pip freeze > requirements.txt`

   Para instalar as dependências, use:
   `pip install -r requirements.txt`

5. **Executar o Projeto**
   Com a estrutura criada e o ambiente virtual ativado, inicie o servidor Flask:
   `python app.py`

Resumo dos Comandos Jinja2 Utilizados

- **Variáveis**: `{{ usuario.nome }}`
- **Condicionais**: `{% if esta_logado %} ... {% endif %}`
- **Listas**: `{% for item in items %} ... {% endfor %}`
- **Dicionários**: `{{ dado.nome }}`
- **Filtros**: `{{ usuario.nome | upper }}`
- **O que acontece nesta linha de comando**:
  `return render_template('index.html', **contexto)`
  A. **contexto Dicionário**: O dicionário contém as variáveis que você deseja disponibilizar no template `index.html`.
  B. **Desempacotamento**: O operador **`**`** desempacota o dicionário em pares chave-valor, assim:
  `usuario=usuario`
  `items=items`
  `esta_logado=esta_logado`
  C. **Passagem para o Template**: O `render_template` passa essas variáveis para o template `index.html`. Assim, no template, você pode usar `{{ usuario.nome }}`, `{{ items }}`, e `{{ esta_logado }}` para acessar os dados.

Resumo

- **Jinja2** é um mecanismo de templates poderoso usado pelo Flask para gerar HTML dinâmico.
- **Funcionalidades**: Permite o uso de variáveis, condicionais, loops, filtros, e herança de templates.
- **Integração com Flask**: **`render_template`** do Flask usa **Jinja2** para processar e renderizar os templates.

O que são Web Forms

Web Forms são formulários interativos usados em páginas da web para coletar informações dos usuários. Eles são fundamentais para a interação entre o usuário e o servidor, permitindo a entrada de dados que podem ser processados, armazenados ou utilizados para diversos fins, como:

- **Registro de Usuários**: Para criar contas em websites.
- **Login**: Para autenticação de usuários.
- **Pesquisa**: Para coletar feedback ou opiniões.
- **Envio de Mensagens**: Como em formulários de contato.
- **Compras Online**: Para detalhes de pagamento e envio.

Componentes de Web Forms

Web Forms normalmente incluem os seguintes componentes:

1. **Campos de Entrada (Input Fields)**:
    - **<input type="text">**: Para entradas de texto simples.
    - **<input type="email">**: Para entradas de e-mail.
    - **<input type="password">**: Para entradas de senha (esconde o texto).
    - **<input type="number">**: Para entradas numéricas.
    - **<textarea>**: Para entradas de múltiplas linhas de texto.
2. **Botões de Envio**:
    - **<input type="submit">** ou **<button type="submit">**: Para enviar os dados do formulário ao servidor.
3. **Checkboxes e Radio Buttons**:
    - **<input type="checkbox">**: Para selecionar múltiplas opções.
    - **<input type="radio">**: Para selecionar uma única opção de um grupo.
4. **Menus de Seleção**:
    - **<select>**: Para selecionar uma opção de uma lista suspensa.
5. **Rótulos (Labels)**:
    - **<label>**: Para associar textos descritivos aos campos de entrada.

Funcionamento de Web Forms

1. **Exibição e Preenchimento**:
    - Um formulário é exibido ao usuário em uma página web. Ele contém campos que o usuário pode preencher com informações.
2. **Envio dos Dados**:
    - Quando o usuário clica em "Enviar", os dados inseridos são enviados ao servidor.
    - O envio pode ser feito via métodos HTTP como GET (dados anexados à URL) ou POST (dados enviados no corpo da requisição).
3. **Processamento**:
    - No servidor, os dados podem ser processados: validados, armazenados em um banco de dados, ou usados para algum tipo de resposta.
4. **Resposta**:
    - O servidor envia uma resposta de volta ao navegador, que pode ser uma nova página, uma mensagem de confirmação, ou outra ação.

Validação
Validação é o processo de garantir que os dados inseridos pelo usuário estão corretos antes de enviá-los ao servidor. Existem dois tipos de validação:

- Validação do Lado do Cliente: Feita no navegador usando JavaScript ou HTML5.
- Validação do Lado do Servidor: Feita no servidor após os dados serem enviados.

Exemplos de Uso de Web Forms

- **Páginas de Login e Cadastro**: Capturam credenciais e dados do usuário.
- **Formulários de Pesquisa**: Coletam respostas de usuários para análise.
- **Formulários de Contato**: Permitem que os usuários enviem mensagens ou consultas ao proprietário do site.
- **Formulários de Pagamento**: Coletam informações de pagamento para transações online.

Web Forms são uma parte essencial do desenvolvimento web, proporcionando uma maneira eficiente e organizada de coletar e processar informações dos usuários.

Exemplo Prático

1. **Instalando o Flask**
   Se você ainda não instalou o Flask, pode fazer isso usando pip:
   ```
   pip install Flask
   ```

2. **Estrutura do Projeto**
   Primeiro, crie uma estrutura básica para o seu projeto Flask:
   ```
   flask_web_forms/
   │
   ├── app.py
   ├── templates/
   │   └── formulario.html
   └── static/
       └── styles.css
   ```

3. **Criando o Arquivo app.py**
   Este arquivo será o núcleo da sua aplicação Flask. Vamos começar importando o Flask e outras bibliotecas necessárias.
   ```
   from flask import Flask, render_template, request, redirect, url_for

   app = Flask(__name__)

   @app.route('/', methods=['GET', 'POST'])
   def index():
       if request.method == 'POST':
           nome = request.form['nome']
           email = request.form['email']
           idade = request.form['idade']
           mensagem = request.form['mensagem']

           # Aqui você pode processar os dados do formulário,
           #como salvar em um banco de dados ou enviar um e-mail.
                  print(f"Nome: {nome}, Email: {email}, Idade: {idade}, Mensagem:
   {mensagem}")

           return redirect(url_for('obrigado'))
   ```

```
    return render_template('formulario.html')

@app.route('/obrigado')
def obrigado():
    return "Obrigado por enviar o formulário!"

if __name__ == '__main__':
    app.run(debug=True)
```

4. **Criando o Template formulario.html**
   Agora, crie um arquivo HTML para renderizar o formulário. Coloque este arquivo na pasta **templates**.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Formulário Flask</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <h2>Preencha o Formulário</h2>
    <form method="POST" action="/">
        <label for="nome">Nome:</label><br>
        <input type="text" id="nome" name="nome" required><br><br>

        <label for="email">Email:</label><br>
        <input type="email" id="email" name="email" required><br><br>

        <label for="idade">Idade:</label><br>
        <input type="number" id="idade" name="idade" required><br><br>

        <label for="mensagem">Mensagem:</label><br>
                            <textarea   id="mensagem"   name="mensagem"   rows="4"
required></textarea><br><br>

        <input type="submit" value="Enviar">
    </form>
</body>
</html>
```

5. **Estilos CSS**
   Você também pode adicionar um arquivo CSS simples para melhorar a aparência do formulário. Crie um arquivo **styles.css** na pasta **static**.

```
body {
    font-family: Arial, sans-serif;
    margin: 20px;
}


h2 {
    color: #333;
    text-align: center;
}

form {
    max-width: 500px; /* Limita a largura máxima do formulário */
    margin: 0 auto;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
```

```
    background-color: #f9f9f9;
}

label {
    font-weight: bold;
    display: block;
    margin-bottom: 5px;
}

input, textarea {
    width: 100%; /* O input e o textarea ocupam 100% da largura disponível */
    padding: 8px;
    margin-bottom: 15px;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box; /* Inclui padding e bordas na largura total */
}

input[type="submit"] {
    background-color: #4CAF50;
    color: white;
    border: none;
    cursor: pointer;
    padding: 10px 15px;
    width: 100%; /* Botão de envio também ocupa 100% da largura */
}

input[type="submit"]:hover {
    background-color: #45a049;
}

ul {
    list-style-type: none;
    padding: 0;
    margin: 20px 0;
    color: red;
    text-align: center;
}
```

6. **Executando o Projeto**
   Para rodar o projeto, use o comando:
   `python app.py`

7. Explicação do Código
   - **Rota index**: Define a rota principal / onde o formulário é renderizado e processado. Se o método de solicitação for POST, os dados do formulário são capturados e processados.
   - **Rota obrigado**: Redireciona o usuário para uma página de agradecimento após o envio do formulário.
   - **Template formulario.html**: Contém o HTML do formulário com vários campos: name, email, age, e message.
   - **Estilos CSS**: Define estilos básicos para o formulário.

8. **Expandindo o Projeto**
   Você pode expandir este exemplo para incluir mais campos de formulário, validação de dados, e processamento mais complexo, como salvar as informações em um banco de dados ou enviar um e-mail de confirmação.

O que é Flask-WTF

É uma extensão para o Flask que facilita a criação e a manipulação de formulários web, integrando o Flask com o WTForms, uma biblioteca popular para a criação e validação de formulários em Python. Ele fornece uma maneira mais simples e intuitiva de trabalhar com formulários, além de adicionar recursos adicionais específicos para o Flask.

Principais Recursos do Flask-WTF

1. **Integração com WTForms**: O flask_wtf integra o Flask com o WTForms, permitindo o uso de suas funcionalidades, como validação e renderização de formulários, de maneira mais fluida.
2. **Validação de Formulários**: Permite validar dados de formulários usando uma série de validadores predefinidos e personalizados. Isso inclui validação de campos obrigatórios, comprimento mínimo e máximo, formato de e-mail, entre outros.
3. **Proteção contra CSRF**: Inclui proteção contra Cross-Site Request Forgery (CSRF) de forma automática. Isso é feito através da geração e verificação de tokens CSRF em formulários, aumentando a segurança das aplicações web.
4. **Gerenciamento de Erros**: Facilita a exibição de mensagens de erro personalizadas e a gestão de erros de validação dos campos do formulário.
5. **Facilidade de Uso**: Simplifica o processo de criação de formulários, incluindo a configuração de campos e suas validações, e a renderização em templates HTML.

Web Forms com Flask-WTF

O Flask-WTF é uma extensão do Flask que facilita a criação e validação de formulários usando o poder do WTForms. Vamos modificar o exemplo anterior para usar Flask-WTF.

1. Instalando Flask-WTF
   Primeiro, instale o Flask-WTF:
   ```
   pip install flask-wtf
   ```

2. **Estrutura do Projeto**
   A estrutura do projeto permanece a mesma:
   ```
   flask_web_forms_wtf/
   │
   ├── app.py
   ├── templates/
   │   └── formulario.html
   │   └── obrigado.html
   └── static/
       └── styles.css
   ```

3. **Criando o Arquivo app.py com Flask-WTF**
   Agora, vamos criar o arquivo **app.py** usando Flask-WTF para gerenciar o formulário:

```
from flask import Flask, render_template, request, redirect, url_for, session
from flask_wtf import FlaskForm
from wtforms import StringField, IntegerField, TextAreaField, SubmitField
from wtforms.validators import DataRequired, Email

app = Flask(__name__)
app.config['SECRET_KEY'] = 'chave_secreta'

class ContatoForm(FlaskForm):
    nome = StringField('Nome', validators=[DataRequired()])
    email = StringField('Email', validators=[DataRequired(), Email()])
    idade = IntegerField('Idade', validators=[DataRequired()])
    mensagem = TextAreaField('Mensagem', validators=[DataRequired()])
    enviar = SubmitField('Enviar')

@app.route('/', methods=['GET', 'POST'])
def index():
    formulario = ContatoForm()
    if formulario.validate_on_submit():
        contato = {
            'nome': formulario.nome.data,
            'email': formulario.email.data,
            'idade': formulario.idade.data,
            'mensagem': formulario.mensagem.data
        }

        session['contato'] = contato # Armazena os dados na sessão

        return redirect(url_for('obrigado'))

    return render_template('formulario.html', formulario=formulario)

@app.route('/obrigado')
def obrigado():
    contato = session.get('contato', None)  # Recupera os dados da sessão
    if contato is None:
        # Redireciona para a página inicial se não houver dados
        return redirect(url_for('index'))
    # Limpa a sessão após o uso
    session.clear()
    return render_template('obrigado.html', contato=contato)

if __name__ == '__main__':
    app.run(debug=True)
```

4. **Criando o Template formulario.html com Flask-WTF**
   Atualize o arquivo **formulario.html** para renderizar o formulário usando os métodos do Flask-WTF:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Formulário Flask</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
```

```
<body>
    <h2>Preencha o Formulário</h2>
    <form method="POST" action="/">
        {{ formulario.hidden_tag() }}
        <div>
            {{ formulario.nome.label }}<br>
            {{ formulario.nome(size=32) }}<br>
        </div>
        <div>
            {{ formulario.email.label }}<br>
            {{ formulario.email(size=32) }}<br>
        </div>
        <div>
            {{ formulario.idade.label }}<br>
            {{ formulario.idade() }}<br>
        </div>
        <div>
            {{ formulario.mensagem.label }}<br>
            {{ formulario.mensagem(rows=4, cols=40) }}<br>
        </div>
        <div>
            {{ formulario.enviar() }}
        </div>
    </form>
</body>
</html>
```

5. Criando o Template obrigado.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Obrigado</title>
</head>
<body>
    <h1>Obrigado!</h1>
    <p>Nome: {{ contato['nome'] | upper }}</p>
    <p>Email: {{ contato['email'] }}</p>
    <p>Idade: {{ contato['idade'] }}</p>
    <p>Mensagem: {{ contato['mensagem'] }}</p>
    <!-- <a href="{{ url_for('index') }}">Voltar</a> -->
    <button onclick="window.location.href='{{ url_for('index') }}'">Voltar</button>
</body>
</html>
```

6. Explicação do Código

- **FlaskForm**: A classe **ContatoForm** herda de **FlaskForm**, e usamos os campos de formulário fornecidos pelo WTForms como **StringField**, **IntegerField**, e **TextAreaField**. Os validadores são usados para garantir que os campos sejam preenchidos corretamente.
- **validate_on_submit**: Verifica se o formulário foi enviado e se todos os campos passaram nas validações.
- **formulario.hidden_tag()**: Gera o campo oculto necessário para a proteção CSRF.
- **Renderização no template**: Cada campo é renderizado usando o método correspondente (formulario.nome, formulario.email, etc.), e as mensagens de validação podem ser exibidas automaticamente.

- **Sessão (session)**:
    - A sessão é usada para armazenar temporariamente o dicionário contato entre as requisições.
    - **app.secret_key** é necessário para proteger a sessão. Ele deve ser um valor seguro e único.
- **Rota / (Método POST)**:
    - Quando o formulário é submetido, os dados do formulário são capturados e armazenados na sessão.
    - Em seguida, o usuário é redirecionado para a view obrigado.
- **Rota `/obrigado`**:
    - Recupera o dicionário **`contato`** da sessão.
    - Se os dados estiverem disponíveis, renderiza o template **`obrigado.html`** com esses dados.
    - **Limpeza da Sessão (`session.clear()`)**:
        - Após recuperar os dados da sessão e antes de renderizar a página obrigado.html, a função session.clear() é chamada. Isso limpa todos os dados armazenados na sessão, garantindo que eles não estarão mais disponíveis após o redirecionamento.
    - Se os dados não estiverem presentes (por exemplo, se o usuário tentar acessar **`/obrigado`** diretamente sem passar pelo formulário), o usuário é redirecionado de volta para a página inicial.

7. **Executando o Projeto**
   Para executar o projeto, use o comando:
   `python app.py`

8. **Benefícios do Flask-WTF**
    - **Validação Simples**: Com Flask-WTF, você pode aplicar validações complexas com facilidade.
    - **Proteção CSRF**: Flask-WTF cuida da proteção CSRF automaticamente, tornando sua aplicação mais segura.
    - **Mensagens de Erro**: O Flask-WTF facilita a exibição de mensagens de erro de validação no template.

Validação de Campo com Flask-WTF

Instale a biblioteca email validator:

```
pip install email_validator
```

Faça o ajuste do **app.py**, de acordo com o código apresentado abaixo:

```
from flask import Flask, render_template, request, redirect, url_for, session
from flask_wtf import FlaskForm
from wtforms import StringField, IntegerField, TextAreaField, SubmitField
from wtforms.validators import DataRequired, Email, InputRequired, NumberRange, Length

app = Flask(__name__)
app.config['SECRET_KEY'] = 'chave_secreta'
```

```
class ContatoForm(FlaskForm):
    nome = StringField(
        'Nome',
        validators=[
            DataRequired(message="O campo de nome é obrigatório."),
            Length(min=2, max=50, message="O nome deve ter entre 2 e 50 caracteres.")
        ]
    )

    email = StringField(
        'Email',
        validators=[
            DataRequired(message="O campo de email é obrigatório."),
            Email(message="Digite um endereço de email válido.")
        ]
    )
    idade = IntegerField(
        'Idade',
        validators=[
            InputRequired(message="O campo de idade é obrigatório."),
            NumberRange(min=0, max=120, message="A idade deve estar entre 0 e 120 anos.")
        ]
    )
    mensagem = TextAreaField(
        'Mensagem',
        validators=[
            DataRequired(message="O campo de mensagem é obrigatório."),
                  Length(min=10,  max=500,  message="A  mensagem  deve  ter  entre  10  e  500
caracteres.")
        ]
    )
    enviar = SubmitField('Enviar')

@app.route('/', methods=['GET', 'POST'])
def index():
    formulario = ContatoForm()
    if formulario.validate_on_submit():
        contato = {
            'nome': formulario.nome.data,
            'email': formulario.email.data,
            'idade': formulario.idade.data,
            'mensagem': formulario.mensagem.data
        }

        session['contato'] = contato # Armazena os dados na sessão

        return redirect(url_for('obrigado'))

    return render_template('formulario.html', formulario=formulario)

@app.route('/obrigado')
def obrigado():
    contato = session.get('contato', None)  # Recupera os dados da sessão
    if contato is None:
        # Redireciona para a página inicial se não houver dados
        return redirect(url_for('index'))
    # Limpa a sessão após o uso
    session.clear()
    return render_template('obrigado.html', contato=contato)

if __name__ == '__main__':
    app.run(debug=True)
```

Explicação do Código

1. **Importações e Configuração**

```
from flask import Flask, render_template, request, redirect, url_for, session
from flask_wtf import FlaskForm
from wtforms import StringField, IntegerField, TextAreaField, SubmitField
from wtforms.validators import DataRequired, Email, InputRequired, NumberRange,
Length

app = Flask(__name__)
app.config['SECRET_KEY'] = 'chave_secreta'
```

- **Flask**: O framework web utilizado para criar o aplicativo.
- **render_template**: Função para renderizar templates HTML.
- **request**: Objeto que contém dados da requisição HTTP.
- **redirect**: Função para redirecionar o usuário para outra página.
- **url_for**: Gera URLs para rotas nomeadas.
- **session**: Objeto para armazenar dados temporários do usuário.
- **FlaskForm**: Classe base para formulários WTForms no Flask.
- **StringField, IntegerField, TextAreaField, SubmitField**: Tipos de campos do formulário.
- **DataRequired, Email, InputRequired, NumberRange, Length**: Validadores para os campos do formulário.
- **SECRET_KEY**: Chave usada pelo Flask para proteger a sessão

2. **Definição do Formulário**

```
class ContatoForm(FlaskForm):
    nome = StringField(
        'Nome',
        validators=[
            DataRequired(message="O campo de nome é obrigatório."),
            Length(min=2, max=50, message="O nome deve ter entre 2 e 50 caracteres.")
        ]
    )
    email = StringField(
        'Email',
        validators=[
            DataRequired(message="O campo de email é obrigatório."),
            Email(message="Digite um endereço de email válido.")
        ]
    )
    idade = IntegerField(
        'Idade',
        validators=[
            InputRequired(message="O campo de idade é obrigatório."),
            NumberRange(min=0, max=120, message="A idade deve estar entre 0 e 120 anos.")
        ]
    )
    mensagem = TextAreaField(
        'Mensagem',
        validators=[
            DataRequired(message="O campo de mensagem é obrigatório."),
            Length(min=10, max=500, message="A mensagem deve ter entre 10 e 500 caracteres.")
        ]
    )
    enviar = SubmitField('Enviar')
```

ContatoForm: Define um formulário de contato com os seguintes campos:

- **nome**: Campo de texto com validação para presença e comprimento.
- **email**: Campo de texto com validação para presença e formato de email.

- **idade**: Campo numérico com validação para presença, intervalo aceitável e formato.
- **mensagem**: Campo de área de texto com validação para presença e comprimento.
- **enviar**: Botão de envio do formulário.

3. **Rotas do Flask**

Rota Principal

```
@app.route('/', methods=['GET', 'POST'])
def index():
    formulario = ContatoForm()
    if formulario.validate_on_submit():
        contato = {
            'nome': formulario.nome.data,
            'email': formulario.email.data,
            'idade': formulario.idade.data,
            'mensagem': formulario.mensagem.data
        }

        session['contato'] = contato  # Armazena os dados na sessão

        return redirect(url_for('obrigado'))

    return render_template('formulario.html', formulario=formulario)
```

- **`@app.route('/')`**: Define a rota para a página inicial.
- **Métodos 'GET' e 'POST'**: Permite que a rota lide com solicitações GET (exibir o formulário) e POST (enviar o formulário).
- **`formulario.validate_on_submit()`**: Verifica se o formulário foi enviado e se os dados são válidos.
- **`contato`**: Cria um dicionário com os dados do formulário.
- **`session['contato']`**: Armazena os dados na sessão para uso posterior.
- **`redirect(url_for('obrigado'))`**: Redireciona para a rota obrigado após a validação bem-sucedida.
- **`render_template('formulario.html', formulario=formulario)`**: Renderiza o template do formulário.

Rota de Obrigado

```
@app.route('/obrigado')
def obrigado():
    contato = session.get('contato', None)  # Recupera os dados da sessão
    if contato is None:
        # Redireciona para a página inicial se não houver dados
        return redirect(url_for('index'))
    # Limpa a sessão após o uso
    session.clear()
    return render_template('obrigado.html', contato=contato)
```

- **`@app.route('/obrigado')`**: Define a rota para a página de agradecimento.
- **`session.get('contato', None)`**: Recupera os dados armazenados na sessão.
- **`redirect(url_for('index'))`**: Redireciona de volta para a página inicial se não houver dados na sessão.
- **`session.clear()`**: Limpa todos os dados da sessão após o uso.
- **`render_template('obrigado.html', contato=contato)`**: Renderiza o template de agradecimento com os dados do formulário.

4. **Execução do Aplicativo**

```
if __name__ == '__main__':
    app.run(debug=True)
```

- `app.run(debug=True)`: Inicia o servidor Flask em modo de depuração, o que permite ver mensagens de erro detalhadas e recarregar o servidor automaticamente ao fazer alterações no código.

Agora faça os ajustes no **formulario.html**, de acordo com o código apresentado abaixo:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Formulário Flask</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
    <script>
        document.addEventListener('DOMContentLoaded', function() {
            const form = document.querySelector('form');
            const inputs = form.querySelectorAll('input, textarea');

            inputs.forEach(input => {
                input.addEventListener('input', function() {
                    const errorDiv = this.closest('div').querySelector('.error');
                    if (errorDiv) {
                        // Verifica se o campo está válido e limpa as mensagens de erro
                        if (this.value.trim() !== '') {
                            errorDiv.innerHTML = ''; // Limpa as mensagens de erro
                            errorDiv.style.display = 'none'; // Oculta a caixa de erro
                        }
                    }
                });
            });
        });
    </script>
</head>
<body>
    <h2>Preencha o Formulário</h2>
    <form method="POST" action="/" novalidate>
        {{ formulario.hidden_tag() }}
        <div>
            {{ formulario.nome.label }}<br>
            {{ formulario.nome(size=32) }}<br>
            {% if formulario.nome.errors %}
                <div class="error">
                    <ul>
                        {% for error in formulario.nome.errors %}
                            <li>{{ error }}</li>
```

```
                    {% endfor %}
                </ul>
            </div>
        {% endif %}
    </div>
    <div>
        {{ formulario.email.label }}<br>
        {{ formulario.email(size=32) }}<br>
        {% if formulario.email.errors %}
            <div class="error">
                <ul>
                    {% for error in formulario.email.errors %}
                        <li>{{ error }}</li>
                    {% endfor %}
                </ul>
            </div>
        {% endif %}
    </div>
    <div>
        {{ formulario.idade.label }}<br>
        {{ formulario.idade() }}<br>
        {% if formulario.idade.errors %}
            <div class="error">
                <ul>
                    {% for error in formulario.idade.errors %}
                        <li>{{ error }}</li>
                    {% endfor %}
                </ul>
            </div>
        {% endif %}
    </div>
    <div>
        {{ formulario.mensagem.label }}<br>
        {{ formulario.mensagem(rows=4, cols=40) }}<br>
        {% if formulario.mensagem.errors %}
            <div class="error">
                <ul>
                    {% for error in formulario.mensagem.errors %}
                        <li>{{ error }}</li>
                    {% endfor %}
                </ul>
            </div>
        {% endif %}
    </div>
    <div>
        {{ formulario.enviar() }}
    </div>
</form>
</body>
</html>
```

Explicação do Código

1. Cabeçalho (Head)

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Formulário Flask</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
    <script>
        document.addEventListener('DOMContentLoaded', function() {
            const form = document.querySelector('form');
            const inputs = form.querySelectorAll('input, textarea');
```

```
            inputs.forEach(input => {
                input.addEventListener('input', function() {
                    const errorDiv = this.closest('div').querySelector('.error');
                    if (errorDiv) {
                            // Verifica se o campo está válido e limpa as mensagens de
erro
                        if (this.value.trim() !== '') {
                            errorDiv.innerHTML = ''; // Limpa as mensagens de erro
                            errorDiv.style.display = 'none'; // Oculta a caixa de erro
                        }
                    }
                });
            });
        });
    </script>
</head>
```

- **<meta charset="UTF-8">**: Define a codificação de caracteres para o documento como UTF-8, que é um padrão para suportar uma ampla gama de caracteres.
- **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Garante que a página se adapta corretamente a diferentes tamanhos de tela, especialmente em dispositivos móveis.
- **<title>Formulário Flask</title>**: Define o título da página que aparece na aba do navegador.
- **<link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">**: Inclui um arquivo CSS para estilizar a página. O `url_for('static', filename='styles.css')` gera o URL para o arquivo CSS armazenado na pasta static.
- Script JavaScript:
  - Adiciona um evento que é acionado quando o conteúdo da página é carregado.
  - Seleciona todos os campos de entrada (`input`) e áreas de texto (`textarea`) do formulário.
  - Adiciona um evento de entrada (input) para limpar e ocultar mensagens de erro quando o usuário começa a digitar.

2. Corpo (Body)

```
<body>
    <h2>Preencha o Formulário</h2>
    <form method="POST" action="/" novalidate>
        {{ formulario.hidden_tag() }}


        <!-- Campos do formulário -->
        <div>
            {{ formulario.nome.label }}<br>
            {{ formulario.nome(size=32) }}<br>
            {% if formulario.nome.errors %}
                <div class="error">
                    <ul>
                        {% for error in formulario.nome.errors %}
                            <li>{{ error }}</li>
                        {% endfor %}
                    </ul>
```

```
                </div>
            {% endif %}
        </div>
        <div>
            {{ formulario.email.label }}<br>
            {{ formulario.email(size=32) }}<br>
            {% if formulario.email.errors %}
                <div class="error">
                    <ul>
                        {% for error in formulario.email.errors %}
                            <li>{{ error }}</li>
                        {% endfor %}
                    </ul>
                </div>
            {% endif %}
        </div>
        <div>
            {{ formulario.idade.label }}<br>
            {{ formulario.idade() }}<br>
            {% if formulario.idade.errors %}
                <div class="error">
                    <ul>
                        {% for error in formulario.idade.errors %}
                            <li>{{ error }}</li>
                        {% endfor %}
                    </ul>
                </div>
            {% endif %}
        </div>
        <div>
            {{ formulario.mensagem.label }}<br>
            {{ formulario.mensagem(rows=4, cols=40) }}<br>
            {% if formulario.mensagem.errors %}
                <div class="error">
                    <ul>
                        {% for error in formulario.mensagem.errors %}
                            <li>{{ error }}</li>
                        {% endfor %}
                    </ul>
                </div>
            {% endif %}
        </div>
        <div>{{ formulario.enviar() }}</div>
    </form>
</body>
```

- **`<h2>Preencha o Formulário</h2>`**: Título da página para orientar o usuário.
- **`<form method="POST" action="/" novalidate>`**: Define o início do formulário, que usa o método POST para enviar dados para a URL raiz (/). O atributo novalidate desativa a validação HTML5 padrão do navegador, permitindo que apenas a validação feita pelo Flask-WTF seja usada.
- **`{{ formulario.hidden_tag() }}`**: Inclui um campo oculto gerado pelo Flask-WTF que é usado para proteção contra CSRF (Cross-Site Request Forgery).
- **Campos do Formulário**:

    - Cada campo é renderizado com sua etiqueta ({{ formulario.nome.label }}) e campo de entrada (`{{ formulario.nome(size=32) }}`).
    - As mensagens de erro são exibidas abaixo de cada campo se houver erros de validação.
    - Div **`.error`**: Contém uma lista não ordenada (<ul>) com erros para cada campo, que são exibidos se o campo tiver erros.
- **`{{ formulario.enviar() }}`**: Renderiza o botão de envio do formulário.

Carga (upload) de Arquivos

1. **Instalação do Flask**
   Se você ainda não tem o Flask instalado, pode instalá-lo usando o pip:
   ```
   pip install flask
   ```

2. **Estrutura do Projeto**
   Vamos criar a seguinte estrutura de pastas para o projeto:
   ```
   upload_app/
   │
   ├── app.py
   ├── templates/
   │   └── upload.html
   └── uploads/
   ```
   - **`app.py`**: Contém o código principal da aplicação Flask.
   - **`templates/upload.html`**: Página HTML para o upload do arquivo.
   - **`uploads/`**: Pasta onde os arquivos enviados serão armazenados.

3. **Código do Flask**
   No arquivo **`app.py`**, adicione o seguinte código:
   ```
   from flask import Flask, render_template, request, redirect, url_for, flash
   import os

   app = Flask(__name__)
   app.secret_key = 'chave_secreta'

   # Defina o caminho absoluto para a pasta de uploads
   upload_folder = r'C:\Dev\SENAI\DSW\003_flask_upload\uploads'
   app.config['UPLOAD_FOLDER'] = upload_folder
   app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024  # Limite de 16 MB para os arquivos

   # Certifique-se de que a pasta de upload existe
   if not os.path.exists(app.config['UPLOAD_FOLDER']):
       os.makedirs(app.config['UPLOAD_FOLDER'])


   @app.route('/')
   def index():
       return render_template('upload.html')


   @app.route('/upload', methods=['POST'])
   def upload_file():
       if 'arquivo' not in request.files:
           flash('Nenhum arquivo foi selecionado')
           return redirect(url_for('index'))

       arquivo = request.files['arquivo']

       if arquivo.filename == '':
           flash('Nenhum arquivo foi selecionado')
           return redirect(url_for('index'))
   ```

```
    if arquivo:
        nome_arquivo = arquivo.filename
        caminho_arquivo = os.path.join(app.config['UPLOAD_FOLDER'], nome_arquivo)
        try:
            arquivo.save(caminho_arquivo)
            flash(f'Arquivo {nome_arquivo} enviado com sucesso!')
        except Exception as e:
            flash(f'Erro ao salvar o arquivo: {e}')
        return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(debug=True)
```

4. **Página HTML**
   No arquivo **`templates/upload.html`**, adicione o seguinte código HTML:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Carga de Arquivo</title>
</head>
<body>
    <h1>Carga de Arquivo</h1>

    {% with messages = get_flashed_messages() %}
      {% if messages %}
        <ul>
        {% for message in messages %}
          <li>{{ message }}</li>
        {% endfor %}
        </ul>
      {% endif %}
    {% endwith %}

    <form method="POST" action="/upload" enctype="multipart/form-data">
        <input type="file" name="arquivo">
        <input type="submit" value="Enviar">
    </form>
</body>
</html>
```

5. **Executando a Aplicação**
   Execute a aplicação com o comando:
   `python app.py`

6. **Testando o Upload**
   Abra o navegador e acesse **`http://localhost:5000`**. Você verá a página de upload de arquivo. Selecione um arquivo e clique em "Enviar". O arquivo será salvo na pasta uploads/ e uma mensagem de sucesso será exibida.

7. **Considerações Finais**
   Este é um exemplo básico. Em produção, é importante validar o tipo de arquivo, renomear arquivos para evitar conflitos, e lidar com outros aspectos de segurança. Além disso, você pode configurar limites de tamanho de arquivo e definir permissões na pasta de upload.

CRUD em Flask

CRUD é um acrônimo que representa as quatro operações básicas que podem ser realizadas em um banco de dados ou em um sistema de gerenciamento de dados. Essas operações são:

- **Create (Criar)**: Adicionar novos registros ou dados ao banco de dados. Em uma aplicação web, isso pode ser feito por meio de um formulário de entrada onde o usuário fornece informações para criar um novo item.
- **Read (Ler)**: Recuperar e visualizar dados existentes no banco de dados. Essa operação permite que os dados sejam consultados e apresentados ao usuário, geralmente em uma lista ou detalhes de um item específico.
- **Update (Atualizar)**: Modificar dados existentes no banco de dados. Isso envolve alterar informações em registros já existentes com base em novas informações fornecidas pelo usuário.
- **Delete (Excluir)**: Remover registros ou dados do banco de dados. Essa operação permite que itens ou dados desnecessários sejam eliminados.

Exemplo Prático
Se você estiver construindo uma aplicação de gerenciamento de tarefas:

- **Create**: Adicionar uma nova tarefa à lista.
- **Read**: Exibir a lista de tarefas ou detalhes de uma tarefa específica.
- **Update**: Alterar o título ou a descrição de uma tarefa existente.
- **Delete**: Remover uma tarefa da lista.

Essas operações são fundamentais para a maioria das aplicações que manipulam dados, seja em bancos de dados relacionais, sistemas de arquivos ou outras formas de armazenamento. O conceito de CRUD ajuda a estruturar e organizar o acesso e a manipulação dos dados de maneira eficiente e consistente.

CRUD Simples (com uso de lista)

Requisitos
Certifique-se de ter o Flask instalado. Você pode instalar com:

```
pip install Flask
```

Estrutura do Projeto
Vamos organizar o projeto da seguinte forma:

```
crud_flask/
    app.py
    templates/
        index.html
        edicao.html
    static/
        style.css
```

Passo 1: Configuração do Projeto

Crie um arquivo chamado **`app.py`** para definir a aplicação Flask:

```
from flask import Flask, render_template, request, redirect, url_for

app = Flask(__name__)

# Simulação de um banco de dados com uma lista
tarefas = []

@app.route('/')
def index():
    return render_template('index.html', tarefas=tarefas)

@app.route('/adicionar', methods=['POST'])
def adicionar():
    tarefa = request.form.get('tarefa')
    if tarefa:
        tarefas.append(tarefa)
    return redirect(url_for('index'))

@app.route('/editar/<int:tarefa_id>', methods=['GET', 'POST'])
def editar(tarefa_id):
    if request.method == 'POST':
        tarefa = request.form.get('tarefa')
        if tarefa:
            tarefas[tarefa_id] = tarefa
        return redirect(url_for('index'))
    tarefa = tarefas[tarefa_id]
    return render_template('edicao.html', tarefa=tarefa, tarefa_id=tarefa_id)

@app.route('/excluir/<int:tarefa_id>')
def excluir(tarefa_id):
    if 0 <= tarefa_id < len(tarefas):
        tarefas.pop(tarefa_id)
    return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(debug=True)
```

Passo 2: Criar Templates HTML

Crie o diretório **`templates`** e adicione dois arquivos: **`index.html`** e **`edicao.html`**.

`index.html`

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Lista de Tarefas</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <h1>Lista de Tarefas</h1>
    <form action="{{ url_for('adicionar') }}" method="post">
        <input type="text" name="tarefa" placeholder="Adicionar nova tarefa" required>
        <button type="submit">Adicionar</button>
    </form>
```

```
    <ul>
        {% for tarefa in tarefas %}
            <li>
                {{ tarefa }}
                <a href="{{ url_for('editar', tarefa_id=loop.index0) }}">Editar</a>
                <a href="{{ url_for('excluir', tarefa_id=loop.index0) }}">Excluir</a>
            </li>
        {% endfor %}
    </ul>
</body>
</html>
```

edicao.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Editar Tarefa</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <h1>Editar Tarefa</h1>
    <form action="{{ url_for('editar', tarefa_id=tarefa_id) }}" method="post">
        <input type="text" name="tarefa" value="{{ tarefa }}" required>
        <button type="submit">Salvar</button>
    </form>
    <a href="{{ url_for('index') }}">Cancelar</a>
</body>
</html>
```

Passo 3: Adicionar Estilo CSS (Opcional)

Crie o diretório **`static`** e adicione um arquivo **`style.css`** para estilizar a aplicação:

```
body {
    font-family: Arial, sans-serif;
    margin: 20px;
}

h1 {
    color: #333;
}


form {
    margin-bottom: 20px;
}


input[type="text"] {
    padding: 10px;
    margin-right: 10px;
}
```

```
button {
    padding: 10px 20px;
    background-color: #007bff;
    color: white;
    border: none;
    cursor: pointer;
}

button:hover {
    background-color: #0056b3;
}

ul {
    list-style-type: none;
    padding: 0;
}

li {
    margin: 10px 0;
}

a {
    margin-left: 10px;
    color: #007bff;
    text-decoration: none;
}

a:hover {
    text-decoration: underline;
}
```

Passo 4: Executar a Aplicação
No terminal, navegue até o diretório do projeto e execute:
`python app.py`

Explicação
Create (Adicionar): O formulário no index.html envia uma nova tarefa para a rota **`/adicionar`**.
Read (Ler): As tarefas são exibidas em uma lista na página principal (`index.html`).
Update (Editar): A rota **`/editar/<tarefa_id>`** exibe um formulário para editar a tarefa existente.
Delete (Excluir): A rota **`/excluir/<tarefa_id>`** remove a tarefa da lista.

CRUD com SQL (Select, Insert, Update e Delete)

Para modificar o CRUD em Flask para usar SQL puro em vez de uma lista, precisamos:

1. Configurar uma conexão com o banco de dados.
2. Executar comandos SQL para inserir, atualizar, excluir e selecionar dados.
3. Adaptar as rotas e as funções para interagir com o banco de dados.

Passo 1: Configurar a Conexão com o Banco de Dados
Vamos usar o SQLite como banco de dados para este exemplo, pois ele é fácil de configurar e não requer instalação adicional. Primeiro, crie um banco de dados e uma tabela para armazenar as tarefas.

criar_db.py:

```
import sqlite3

# Cria a tabela no banco de dados (execute isso uma vez para configurar o banco de dados)
def init_db():
    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS tarefas (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            descricao TEXT NOT NULL
        )
    ''')
    conn.commit()
    conn.close()

init_db()  # Inicializa o banco de dados na primeira execução
```

Passo 2: Adaptar o `app.py` para Usar SQL
Aqui está a versão atualizada do seu arquivo **`app.py`**, que agora utiliza SQL para manipular as tarefas:

```
from flask import Flask, render_template, request, redirect, url_for
import sqlite3

app = Flask(__name__)


# Função para conectar ao banco de dados
def get_db_connection():
    conn = sqlite3.connect('database.db')
    conn.row_factory = sqlite3.Row
    return conn


@app.route('/')
def index():
    conn = get_db_connection()
    tarefas = conn.execute('SELECT * FROM tarefas').fetchall()
    conn.close()
    return render_template('index.html', tarefas=tarefas)
```

```
@app.route('/adicionar', methods=['POST'])
def adicionar():
    descricao = request.form.get('tarefa')
    if descricao:
        conn = get_db_connection()
        conn.execute('INSERT INTO tarefas (descricao) VALUES (?)', (descricao,))
        conn.commit()
        conn.close()
    return redirect(url_for('index'))

@app.route('/editar/<int:tarefa_id>', methods=['GET', 'POST'])
def editar(tarefa_id):
    conn = get_db_connection()
    if request.method == 'POST':
        descricao = request.form.get('tarefa')
        if descricao:
                    conn.execute('UPDATE tarefas SET descricao = ? WHERE id = ?',
(descricao, tarefa_id))
            conn.commit()
            conn.close()
            return redirect(url_for('index'))
          tarefa = conn.execute('SELECT * FROM tarefas WHERE id = ?',
(tarefa_id,)).fetchone()
    conn.close()
    return render_template('edicao.html', tarefa=tarefa, tarefa_id=tarefa_id)

@app.route('/excluir/<int:tarefa_id>')
def excluir(tarefa_id):
    conn = get_db_connection()
    conn.execute('DELETE FROM tarefas WHERE id = ?', (tarefa_id,))
    conn.commit()
    conn.close()
    return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(debug=True)
```

Passo 3: Adaptar os Templates HTML

Os templates **`index.html`** e **`edicao.html`** funcionarão quase da mesma forma, mas você precisará acessar os dados de forma ligeiramente diferente, pois agora eles vêm do banco de dados.

`index.html`:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Lista de Tarefas</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <h1>Lista de Tarefas</h1>
    <form action="{{ url_for('adicionar') }}" method="post">
        <input type="text" name="tarefa" placeholder="Adicionar nova tarefa" required>
        <button type="submit">Adicionar</button>
    </form>
```

```
    <ul>
        {% for tarefa in tarefas %}
            <li>
                {{ tarefa['descricao'] }}
                <a href="{{ url_for('editar', tarefa_id=tarefa['id']) }}">Editar</a>
                <a href="{{ url_for('excluir', tarefa_id=tarefa['id']) }}">Excluir</a>
            </li>
        {% endfor %}
    </ul>
</body>
</html>
```

`edicao.html`:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Editar Tarefa</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <h1>Editar Tarefa</h1>
    <form action="{{ url_for('editar', tarefa_id=tarefa_id) }}" method="post">
        <input type="text" name="tarefa" value="{{ tarefa['descricao'] }}" required>
        <button type="submit">Salvar</button>
    </form>
    <a href="{{ url_for('index') }}">Cancelar</a>
</body>
</html>
```

Explicação das Mudanças

- Conexão com o Banco de Dados: Criamos uma função **`get_db_connection()`** para abrir uma conexão com o banco de dados SQLite. Essa função é usada em cada rota que precisa acessar o banco.
- tarefas Agora no Banco de Dados: As tarefas agora são armazenadas e manipuladas usando SQL puro em vez de uma lista em memória.
- Consultas SQL: Utilizamos **`SELECT`**, **`INSERT`**, **`UPDATE`**, e **`DELETE`** para realizar as operações CRUD.

Passo 4: Executar a Aplicação

Após implementar as mudanças, você pode rodar sua aplicação com:

```
python app.py
```

Acesse a aplicação em `http://127.0.0.1:5000/` para ver o CRUD em ação, agora utilizando um banco de dados SQLite.

CRUD com SQLAlchemy

Vamos usar SQLAlchemy para lidar com o banco de dados e suas operações. Abaixo está o código modificado:

1. **Instalar dependências:**
   Certifique-se de ter o SQLAlchemy e o Flask-SQLAlchemy instalados. Você pode instalá-los via pip:
   `pip install sqlalchemy flask-sqlalchemy`

2. **Definir o modelo de dados:**
   Primeiro, vamos definir o modelo de dados que representa a tabela **`tarefas`** no banco de dados.

   app.py:

```
from flask import Flask, render_template, request, redirect, url_for
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

# Configuração do banco de dados
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///bd_tarefas.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

# Modelo da Tarefa
class Tarefa(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    descricao = db.Column(db.String(200), nullable=False)

# Criar o banco de dados e a tabela
with app.app_context():
    db.create_all()

@app.route('/')
def index():
    tarefas = Tarefa.query.all()
    return render_template('index.html', tarefas=tarefas)

@app.route('/adicionar', methods=['POST'])
def adicionar():
    descricao = request.form.get('tarefa')
    if descricao:
        nova_tarefa = Tarefa(descricao=descricao)
        db.session.add(nova_tarefa)
        db.session.commit()
    return redirect(url_for('index'))

@app.route('/editar/<int:tarefa_id>', methods=['GET', 'POST'])
def editar(tarefa_id):
    tarefa = Tarefa.query.get_or_404(tarefa_id)
    if request.method == 'POST':
        descricao = request.form.get('tarefa')
```

```
        if descricao:
            tarefa.descricao = descricao
            db.session.commit()
            return redirect(url_for('index'))
    return render_template('edicao.html', tarefa=tarefa, tarefa_id=tarefa_id)

@app.route('/excluir/<int:tarefa_id>')
def excluir(tarefa_id):
    tarefa = Tarefa.query.get_or_404(tarefa_id)
    db.session.delete(tarefa)
    db.session.commit()
    return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(debug=True)
```

3. **Modificar os templates HTML:**
   A principal modificação nos templates HTML será ajustar a forma como os dados das tarefas são acessados. Em vez de **`tarefa['descricao']`**, agora você acessará **`tarefa.descricao`**.

   index.html:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Lista de Tarefas</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <h1>Lista de Tarefas</h1>
    <form action="{{ url_for('adicionar') }}" method="post">
        <input type="text" name="tarefa" placeholder="Adicionar nova tarefa" required>
        <button type="submit">Adicionar</button>
    </form>
    <ul>
        {% for tarefa in tarefas %}
            <li>
                {{ tarefa.descricao }}
                <a href="{{ url_for('editar', tarefa_id=tarefa.id) }}">Editar</a>
                <a href="{{ url_for('excluir', tarefa_id=tarefa.id) }}">Excluir</a>
            </li>
        {% endfor %}
    </ul>
</body>
</html>
```

   edicao.html:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Editar Tarefa</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
```

```
    <h1>Editar Tarefa</h1>
    <form action="{{ url_for('editar', tarefa_id=tarefa_id) }}" method="post">
        <input type="text" name="tarefa" value="{{ tarefa.descricao }}" required>
        <button type="submit">Salvar</button>
    </form>
    <a href="{{ url_for('index') }}">Cancelar</a>
</body>
</html>
```

4. **Explicação:**
   - **Tarefa**: É a classe do modelo SQLAlchemy que representa a tabela **tarefas**. Cada instância da classe **Tarefa** corresponde a uma linha na tabela.
   - **db.session**: É usado para adicionar, atualizar ou deletar registros no banco de dados.
   - **get_or_404**: Obtém um objeto por seu ID ou retorna um erro 404 se ele não for encontrado.

5. **Rodar o projeto:**
   Com as alterações feitas, você pode rodar o projeto Flask como de costume, utilizando:
   `python app.py`

CRUD com SQLAlchemy e WTF

Para melhorar a criação e a validação de formulários no Flask, podemos utilizar o Flask-WTF, uma extensão que integra o Flask com o WTForms, facilitando a criação de formulários robustos e seguros.

1. **Instalar dependências**
   Primeiro, certifique-se de instalar o Flask-WTF:
   `pip install flask-wtf`

2. **Configuração do Flask-WTF**
   Você precisará configurar uma chave secreta no Flask para usar Flask-WTF, que é necessária para proteger os formulários contra ataques CSRF (Cross-Site Request Forgery).

   app.py:

```
from flask import Flask, render_template, request, redirect, url_for
from flask_sqlalchemy import SQLAlchemy
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField
from wtforms.validators import DataRequired

app = Flask(__name__)

# Configuração do banco de dados
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///bd_tarefas.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SECRET_KEY'] = 'sua_chave_secreta_aqui'    # Substitua por uma chave
secreta


db = SQLAlchemy(app)


# Modelo da Tarefa
class Tarefa(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    descricao = db.Column(db.String(200), nullable=False)


# Criar o banco de dados e a tabela
with app.app_context():
    db.create_all()


# Formulário para adicionar/editar tarefa
class TarefaForm(FlaskForm):
    descricao = StringField('Tarefa', validators=[DataRequired()])
    submit = SubmitField('Salvar')


@app.route('/')
def index():
    tarefas = Tarefa.query.all()
    return render_template('index.html', tarefas=tarefas)
```

```
@app.route('/adicionar', methods=['GET', 'POST'])
def adicionar():
    formulario = TarefaForm()
    if formulario.validate_on_submit():
        nova_tarefa = Tarefa(descricao=formulario.descricao.data)
        db.session.add(nova_tarefa)
        db.session.commit()
        return redirect(url_for('index'))
    return render_template('adicionar.html', formulario=formulario)

@app.route('/editar/<int:tarefa_id>', methods=['GET', 'POST'])
def editar(tarefa_id):
    tarefa = Tarefa.query.get_or_404(tarefa_id)
    formulario = TarefaForm(obj=tarefa)
    if formulario.validate_on_submit():
        tarefa.descricao = formulario.descricao.data
        db.session.commit()
        return redirect(url_for('index'))
                return    render_template('edicao.html',    formulario=formulario,
tarefa_id=tarefa_id)

@app.route('/excluir/<int:tarefa_id>')
def excluir(tarefa_id):
    tarefa = Tarefa.query.get_or_404(tarefa_id)
    db.session.delete(tarefa)
    db.session.commit()
    return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(debug=True)
```

3. **Atualização do CSS**
   Vamos ajustar o style.css para que a visualização dos botões dos formulários de adição e edição tenham uma melhor visualização.

```
/* Estilo do botão padrão */
button, .button-link {
    padding: 10px 20px;
    font-size: 16px;
    background-color: #007bff;
    color: white;
    border: none;
    cursor: pointer;
    text-align: center;
    display: inline-block;
    text-decoration: none;
    line-height: 1.5; /* Adiciona consistência na altura */
}


button:hover, .button-link:hover {
    background-color: #0056b3;
}
```

```
/* Estilo específico para o botão "Cancelar" */
.button-link {
    background-color: #dc3545;
}

.button-link:hover {
    background-color: #c82333;
}
```

4. **Modificar os templates HTML**
   Agora vamos modificar os templates HTML para usar os formulários do Flask-WTF.

   adicionar.html (Novo template)

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Adicionar Tarefa</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <h1>Adicionar Tarefa</h1>
    <form action="{{ url_for('adicionar') }}" method="post">
        {{ formulario.hidden_tag() }}
        <div>
            {{ formulario.descricao.label }}<br>
            {{ formulario.descricao(size=40) }}<br>
            {% for error in formulario.descricao.errors %}
                <span style="color: red;">[{{ error }}]</span><br>
            {% endfor %}
        </div>
        <div>
            {{ formulario.submit(class_='button-link') }}
            <a href="{{ url_for('index') }}" class="button-link">Cancelar</a>
        </div>
    </form>
</body>
</html>
```

   edicao.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Editar Tarefa</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <h1>Editar Tarefa</h1>
    <form action="{{ url_for('editar', tarefa_id=tarefa_id) }}" method="post">
        {{ formulario.hidden_tag() }}
        <div>
            {{ formulario.descricao.label }}<br>
            {{ formulario.descricao(size=40) }}<br>
            {% for error in formulario.descricao.errors %}
                <span style="color: red;">[{{ error }}]</span><br>
```

```
            {% endfor %}
        </div>

        <div>
            {{ formulario.submit(class_='button-link') }}
            <a href="{{ url_for('index') }}" class="button-link">Cancelar</a>
        </div>
    </form>
</body>
</html>
```

index.html (com link para o template adicionar.html)

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Lista de Tarefas</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <h1>Lista de Tarefas</h1>
    <a href="{{ url_for('adicionar') }}">Adicionar Nova Tarefa</a>
    <ul>
        {% for tarefa in tarefas %}
            <li>
                {{ tarefa.descricao }}
                <a href="{{ url_for('editar', tarefa_id=tarefa.id) }}">Editar</a>
                <a href="{{ url_for('excluir', tarefa_id=tarefa.id) }}">Excluir</a>
            </li>
        {% endfor %}
    </ul>
</body>
</html>
```

5. **Explicação**
   - **`TarefaForm`**: Classe que define o formulário de tarefa, utilizando Flask-WTF. Ela possui um campo de texto (**`descricao`**) e um botão de submit (**`submit`**).
   - **`formulario.hidden_tag()`**: Adiciona um campo oculto ao formulário para proteger contra ataques CSRF.
   - **`formulario.validate_on_submit()`**: Valida o formulário quando o método de requisição for POST.
   - **`FlaskForm`**: Base para todos os formulários, fornecendo métodos e validações fáceis de usar.
   - **`Submit Button`**: Continua sendo o botão de envio padrão, mas agora ambos os botões estarão estilizados de forma consistente.
   - **`Cancelar Button`**: O link "Cancelar" agora se parece com um botão vermelho, que se ajusta ao estilo do "Submit".
   - **Padding e Font Size Consistentes**: Definir o **`padding`** e **`font-size`** iguais para ambos os botões faz com que eles tenham a mesma altura.
   - **`Line Height`**: A adição de **`line-height`** ajuda a garantir que o texto dentro dos botões seja centralizado verticalmente, aumentando a consistência visual.

6. **Rodar o projeto**
   Após as alterações, você pode rodar o projeto como antes com:
   `python app.py`

Agora seu CRUD usa Flask-WTF para gerenciar os formulários, garantindo uma camada extra de segurança e facilidade no desenvolvimento de formulários no Flask.

Bootstrap Básico

1. O que é Bootstrap?

Bootstrap é um framework front-end gratuito e open-source que facilita a criação de sites e aplicações web responsivas e modernas. Ele fornece uma coleção de ferramentas de design, incluindo layouts flexíveis, componentes prontos para uso (botões, formulários, navegação, etc.), e utilitários CSS e JavaScript.

2. Configuração do Bootstrap

Há duas maneiras principais de adicionar Bootstrap ao seu projeto: usando um CDN (Content Delivery Network) ou baixando os arquivos diretamente.

Usando CDN

Adicione as seguintes linhas ao <head> do seu documento HTML para incluir o Bootstrap via CDN:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Meu Projeto Bootstrap</title>
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <!-- Seu conteúdo aqui -->
    <!-- Bootstrap JS e Popper.js -->
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.min.js"></script>
</body>
</html>
```

Baixando os Arquivos

Se preferir, você pode baixar o Bootstrap do site oficial (getbootstrap.com) e incluir os arquivos CSS e JS localmente.

3. Criando um Layout Básico

Vamos criar um layout básico com um cabeçalho, uma área de conteúdo e um rodapé.

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Meu Projeto Bootstrap</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <!-- Navbar -->
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">Meu Site</a>
             <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav"
aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav ms-auto">
```

```
                    <li class="nav-item">
                        <a class="nav-link active" aria-current="page" href="#">Início</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#">Sobre</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#">Contato</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>

    <!-- Conteúdo Principal -->
    <div class="container mt-5">
        <div class="row">
            <div class="col-12">
                <h1>Bem-vindo ao Meu Site</h1>
                <p>Este é um exemplo de layout simples usando Bootstrap.</p>
            </div>
        </div>
    </div>

    <!-- Rodapé -->
    <footer class="bg-dark text-white text-center py-3 mt-5">
        <p>© 2024 Meu Site. Todos os direitos reservados.</p>
    </footer>

    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.min.js"></script>
</body>
</html>
```

4. Trabalhando com Grids

O sistema de grid do Bootstrap é baseado em colunas que permitem criar layouts responsivos. Aqui está um exemplo básico:

```
<div class="container">
    <div class="row">
        <div class="col-md-4">
            <div class="p-3 border bg-light">Coluna 1</div>
        </div>
        <div class="col-md-4">
            <div class="p-3 border bg-light">Coluna 2</div>
        </div>
        <div class="col-md-4">
            <div class="p-3 border bg-light">Coluna 3</div>
        </div>
    </div>
</div>
```

Esse layout se ajusta automaticamente em dispositivos de diferentes tamanhos, organizando as colunas conforme necessário.

5. Componentes do Bootstrap

O Bootstrap vem com muitos componentes prontos para uso. Aqui estão alguns exemplos:

Botões

```
<button type="button" class="btn btn-primary">Botão Primário</button>
<button type="button" class="btn btn-secondary">Botão Secundário</button>
```

Cards

```
<div class="card" style="width: 18rem;">
    <img src="https://via.placeholder.com/150" class="card-img-top" alt="...">
    <div class="card-body">
        <h5 class="card-title">Título do Card</h5>
        <p class="card-text">Algum texto de exemplo para este card.</p>
        <a href="#" class="btn btn-primary">Ir para algum lugar</a>
    </div>
</div>
```

Alertas

```
<div class="alert alert-warning" role="alert">
  Este é um alerta de aviso — verifique!
</div>
```

6. Personalizando com CSS

Você pode personalizar o Bootstrap adicionando seu próprio CSS. Basta incluir um arquivo CSS adicional após o Bootstrap:

```
<link href="meu-estilo.css" rel="stylesheet">
```

Dentro do `meu-estilo.css`, você pode sobrescrever qualquer estilo padrão do Bootstrap para atender às necessidades específicas do seu projeto.

Exemplo de uma personalização com CSS

Vamos começar com um botão básico do Bootstrap:

```
<button type="button" class="btn btn-primary">Botão Padrão</button>
```

Por padrão, este botão terá um fundo azul e texto branco, de acordo com as classes **`btn`** e **`btn-primary`** do Bootstrap.

Agora, digamos que você queira mudar a cor de fundo e o texto desse botão. Para fazer isso, você pode criar um arquivo CSS personalizado e sobrescrever as classes do Bootstrap.

Passos:

1. Crie um arquivo CSS personalizado (por exemplo, `meu-estilo.css`).
2. Adicione o CSS no seu HTML após o Bootstrap para garantir que o seu estilo seja aplicado.

index.html

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Exemplo de Sobrescrita de Estilo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
    <link href="meu-estilo.css" rel="stylesheet"> <!-- Adicione seu CSS aqui -->
</head>
<body>
    <div class="container mt-5">
        <button type="button" class="btn btn-primary">Botão Padrão</button>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.min.js"></script>
</body>
</html>
```

meu-estilo.css

```
/* Sobrescrevendo o estilo do botão .btn-primary */
.btn-primary {
    background-color: #ff5733; /* Novo fundo: cor laranja */
    border-color: #ff5733;     /* Nova cor da borda */
    color: #ffffff;            /* Cor do texto: branca */
}

/* Mudando o estilo do botão quando o mouse está sobre ele */
.btn-primary:hover {
    background-color: #c70039; /* Fundo ao passar o mouse: cor vermelha */
    border-color: #c70039;     /* Borda ao passar o mouse */
    color: #ffffff;            /* Cor do texto continua branca */
}
```

Como Funciona a Sobrescrita

- **Ordem de Importação**: O CSS que vem depois na ordem de importação tem maior prioridade. No exemplo, meu-estilo.css foi importado após o Bootstrap, então ele sobrescreve os estilos padrão do Bootstrap.
- **Especificidade**: Se necessário, você pode aumentar a especificidade do seletor no seu CSS para garantir que ele sobrescreva os estilos do Bootstrap. Exemplo:

```
/* Usando mais especificidade */
.container .btn-primary {
    background-color: #ff5733;
    border-color: #ff5733;
    color: #ffffff;
}
```

Depois de adicionar esse CSS, o botão no seu site terá o fundo laranja (`#ff5733`) e o texto branco, ao invés do estilo azul padrão do Bootstrap.

Isso é útil quando você precisa personalizar o visual do Bootstrap para que ele se adapte ao design específico do seu projeto.

7. Referências e Documentação
Para explorar mais recursos e componentes do Bootstrap, visite a documentação oficial.