Semana 1: Introdução ao Flask

- **Dia 1: 30/07/2024**
  - **18:30 - 19:00:** Apresentação do curso e objetivos
  - **19:00 - 20:00:** Introdução ao Flask e suas funcionalidades
  - **20:00 - 21:00:** Configuração do ambiente de desenvolvimento com Flask
  - **21:00 - 22:00:** Primeira aplicação Flask: "Hello World"
- **Dia 2: 01/08/2024**
  - **18:30 - 19:00:** Estrutura de pastas em uma aplicação Flask
  - **19:00 - 20:00:** Rotas e view functions
  - **20:00 - 21:00:** Renderização de templates com Jinja2
  - **21:00 - 22:00:** Exercício prático: Criando uma página web com Flask

Semana 2: Templates e Formulários

- **Dia 3: 06/08/2024**
  - **18:30 - 19:00:** Uso de templates no Flask
  - **19:00 - 20:00:** Passagem de variáveis para templates
  - **20:00 - 21:00:** Criação de layouts com templates
  - **21:00 - 22:00:** Exercício prático: Criando uma página de layout com Flask
- **Dia 4: 08/08/2024**
  - **18:30 - 19:00:** Introdução a formulários no Flask
  - **19:00 - 20:00:** Biblioteca Flask-WTF para formulários
  - **20:00 - 21:00:** Validação de formulários
  - **21:00 - 22:00:** Exercício prático: Criando e validando um formulário com Flask-WTF

Semana 3: Banco de Dados e SQLAlchemy

- **Dia 5: 13/08/2024**
  - **18:30 - 19:00:** Introdução ao SQLAlchemy
  - **19:00 - 20:00:** Configuração do SQLAlchemy no Flask
  - **20:00 - 21:00:** Criação de modelos de banco de dados
  - **21:00 - 22:00:** Exercício prático: Criando e configurando um banco de dados com SQLAlchemy
- **Dia 6: 15/08/2024**
  - **18:30 - 19:00:** Operações CRUD (Create, Read, Update, Delete)
  - **19:00 - 20:00:** Realizando consultas no banco de dados
  - **20:00 - 21:00:** Relacionamentos entre tabelas
  - **21:00 - 22:00:** Exercício prático: Implementando operações CRUD com SQLAlchemy

Semana 4: Autenticação e Autorização

- **Dia 7: 20/08/2024**
  - **18:30 - 19:00:** Introdução à autenticação no Flask
  - **19:00 - 20:00:** Biblioteca Flask-Login
  - **20:00 - 21:00:** Implementação de autenticação de usuários
  - **21:00 - 22:00:** Exercício prático: Criando um sistema de login e registro

- **Dia 8: 22/08/2024**
    - **18:30 - 19:00:** Gerenciamento de sessões de usuários
    - **19:00 - 20:00:** Autorização e controle de acesso
    - **20:00 - 21:00:** Proteção de rotas
    - **21:00 - 22:00:** Exercício prático: Implementando controle de acesso

Semana 5: APIs com Flask

- **Dia 9: 27/08/2024**
    - **18:30 - 19:00:** Introdução a APIs RESTful
    - **19:00 - 20:00:** Criação de uma API com Flask
    - **20:00 - 21:00:** Endpoints e métodos HTTP
    - **21:00 - 22:00:** Exercício prático: Criando uma API básica com Flask
- **Dia 10: 29/08/2024**
    - **18:30 - 19:00:** Manipulação de dados JSON
    - **19:00 - 20:00:** Consumo de APIs externas
    - **20:00 - 21:00:** Autenticação em APIs (JWT)
    - **21:00 - 22:00:** Exercício prático: Implementando autenticação em uma API

Semana 6: Primeira Avaliação

- **Dia 11: 03/09/2024**
    - **18:30 - 19:00:** Revisão dos conteúdos abordados
    - **19:00 - 20:00:** Discussão de dúvidas e exemplos práticos
    - **20:00 - 21:00:** Preparação para a avaliação escrita
    - **21:00 - 22:00:** Avaliação escrita intermediária 1
- **Dia 12: 05/09/2024**
    - **18:30 - 19:00:** Correção e discussão da avaliação
    - **19:00 - 20:00:** Introdução ao projeto final
    - **20:00 - 21:00:** Definição dos requisitos do projeto
    - **21:00 - 22:00:** Formação de grupos e início do planejamento do projeto

Semana 7: Desenvolvimento do Projeto Final (Parte 1)

- **Dias 13 e 14: 10/09/2024 e 12/09/2024**
    - **18:30 - 19:00:** Planejamento e definição das funcionalidades do projeto
    - **19:00 - 20:00:** Design da arquitetura da aplicação
    - **20:00 - 21:00:** Desenvolvimento de funcionalidades iniciais
    - **21:00 - 22:00:** Implementação e testes iniciais

Semana 8: Desenvolvimento do Projeto Final (Parte 2)

- **Dias 15 e 16: 17/09/2024 e 19/09/2024**
    - **18:30 - 19:00:** Desenvolvimento de funcionalidades intermediárias
    - **19:00 - 20:00:** Integração com banco de dados e formulários
    - **20:00 - 21:00:** Implementação de autenticação e autorização
    - **21:00 - 22:00:** Testes e correções

Semana 9: Desenvolvimento do Projeto Final (Parte 3)

- **Dias 17 e 18: 24/09/2024 e 26/09/2024**
    - **18:30 - 19:00:** Desenvolvimento de funcionalidades avançadas
    - **19:00 - 20:00:** Implementação de API
    - **20:00 - 21:00:** Integração de front-end com back-end
    - **21:00 - 22:00:** Testes e validações finais

Semana 10: Segunda Avaliação

- **Dia 19: 01/10/2024**
  - **18:30 - 19:00:** Revisão dos conteúdos abordados
  - **19:00 - 20:00:** Discussão de dúvidas e exemplos práticos
  - **20:00 - 21:00:** Preparação para a avaliação escrita
  - **21:00 - 22:00:** Avaliação escrita intermediária 2
- **Dia 20: 03/10/2024**
  - **18:30 - 19:00:** Correção e discussão da avaliação
  - **19:00 - 20:00:** Ajustes e melhorias no projeto final
  - **20:00 - 21:00:** Preparação para apresentação do projeto
  - **21:00 - 22:00:** Ensaios de apresentação

Semana 11: Finalização do Projeto

- **Dias 21 e 22: 08/10/2024 e 15/10/2024**
  - **18:30 - 19:00:** Revisão final do projeto
  - **19:00 - 20:00:** Correções de bugs e melhorias
  - **20:00 - 21:00:** Documentação do projeto
  - **21:00 - 22:00:** Preparação para a apresentação final

Semana 12: Apresentação do Projeto Final

- **Dias 23 e 24: 22/10/2024 e 29/10/2024**
  - **18:30 - 19:00:** Apresentação dos projetos pelos grupos
  - **19:00 - 20:00:** Feedback dos colegas e do professor
  - **20:00 - 21:00:** Discussão sobre desafios e aprendizados
  - **21:00 - 22:00:** Encerramento da disciplina e entrega das notas

Preparando o ambiente virtual para um projeto Flask

1. Criando um Ambiente Virtual

1.1. Crie um diretório para o seu projeto:

```
mkdir meu_projeto_flask
cd meu_projeto_flask
```

1.2. Crie o ambiente virtual dentro do diretório do projeto:

```
python -m venv .venv
```

O nome .venv é uma das convenções comuns, mas você pode escolher outro nome se preferir.

2. Ativando o Ambiente Virtual

Após criar o ambiente virtual, você precisa ativá-lo:

```
.venv\Scripts\activate
```

Você deve ver o nome do ambiente virtual (por exemplo, (.venv)) antes do prompt de comando, indicando que o ambiente está ativado.

3. Instalando o Flask

Com o ambiente virtual ativado, você pode instalar o Flask e outras dependências necessárias usando o pip:

```
pip install Flask
```

4. Criando um Arquivo requirements.txt

Para gerenciar e compartilhar as dependências do seu projeto, crie um arquivo requirements.txt que lista todos os pacotes instalados. Você pode gerar esse arquivo com o seguinte comando:

```
pip freeze > requirements.txt
```

Isso cria um arquivo requirements.txt com as versões exatas dos pacotes instalados no ambiente virtual.

5. Desativando o Ambiente Virtual

Quando terminar de trabalhar no projeto, você pode desativar o ambiente virtual com o comando:

```
deactivate
```

6. Reativando o Ambiente Virtual

Quando voltar ao projeto, basta reativar o ambiente virtual usando o mesmo comando de ativação descrito acima.

7. Instalando Dependências em Outro Ambiente

Se você precisar instalar as dependências em outro ambiente (por exemplo, em outro computador ou para outro desenvolvedor), você pode usar o arquivo requirements.txt com o comando:

```
pip install -r requirements.txt
```

Dia 1: 30/07/2024

18:30 - 19:00: Apresentação do curso e objetivos

Objetivos do Curso:

- Compreender os fundamentos do desenvolvimento web.
- Aprender a utilizar o framework Flask para criar aplicações web.
- Desenvolver habilidades práticas em programação web com Python.
- Criar e configurar um ambiente de desenvolvimento web.

19:00 - 20:00: Introdução ao Flask e suas funcionalidades

Introdução ao Flask:

- Flask é um microframework para desenvolvimento web em Python.
- É leve e flexível, ideal para pequenas aplicações e protótipos.
- Oferece funcionalidades essenciais, permitindo adicionar outras conforme necessário.

20:00 - 21:00: Configuração do ambiente de desenvolvimento com Flask

Configuração do Ambiente de Desenvolvimento:

1. Instalar Python: Certifique-se de ter o Python instalado. Você pode baixar e instalar a versão mais recente do site oficial do Python.

2. Criar um Ambiente Virtual: Crie um ambiente virtual para isolar as dependências do seu projeto.
   ```
   python -m venv venv
   ```

3. Ativar o Ambiente Virtual:
   ```
   venv\Scripts\activate
   ```

4. Instalar o Flask:
   ```
   pip install Flask
   ```

21:00 - 22:00: Primeira aplicação Flask: "Olá Mundo"

Estrutura do Projeto:

```
meu_app/
│
├── app.py
├── templates/
│   └── index.html
├── static/
│   └── css/
│       └── style.css
└── venv/
```

Código do app.py:

```
from flask import Flask, render_template

app = Flask(__name__)

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

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

Código do templates/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>Olá Mundo</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/style.css') }}">
</head>
<body>
    <h1>Olá Mundo!</h1>
</body>
</html>
```

Código do static/css/style.css:

```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    text-align: center;
    padding-top: 50px;
}

h1 {
    color: #333;
}
```

Explicação e Passos

1. **Criação do Projeto**: Crie a estrutura de pastas conforme descrito.
2. **Código Principal**: No app.py, criamos uma instância do Flask e definimos uma rota básica que renderiza um template HTML.
3. **Templates e Estilos**: O arquivo index.html dentro da pasta templates contém o conteúdo HTML. O CSS está na pasta static/css.

Executando a Aplicação

1. **No Terminal**: Navegue até a pasta do projeto e ative o ambiente virtual.
2. **Rodando a Aplicação**:
   ```
   python app.py
   ```
3. **Acessar no Navegador**: Abra o navegador e vá para http://127.0.0.1:5000/.

Dia 2: 01/08/2024

18:30 - 19:00: Estrutura de pastas em uma aplicação Flask

Estrutura de Pastas:

```
meu_app/
│
├── app.py
├── templates/
│   ├── base.html
│   ├── home.html
│   └── about.html
├── static/
│   ├── css/
│   │   └── style.css
│   ├── js/
│   │   └── script.js
│   └── img/
│       └── logo.png
└── venv/
```

19:00 - 20:00: Rotas e view functions

Rotas e View Functions:

No app.py, definimos diferentes rotas e suas respectivas funções de visualização:

```
from flask import Flask, render_template

app = Flask(__name__)

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

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

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

20:00 - 21:00: Renderização de templates com Jinja2
Templates com Jinja2:

Criamos um template base (base.html) para reutilizar código e outros templates (home.html, about.html) que herdam de base.html.

Código do templates/base.html:

```
<!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 App{% endblock %}</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/style.css') }}">
</head>
<body>
    <header>
        <h1>Meu App</h1>
        <nav>
            <ul>
                <li><a href="{{ url_for('home') }}">Home</a></li>
                <li><a href="{{ url_for('about') }}">About</a></li>
            </ul>
        </nav>
    </header>
    <main>
        {% block content %}{% endblock %}
    </main>
    <script src="{{ url_for('static',
filename='js/script.js') }}"></script>
</body>
</html>
```

Código do templates/home.html:

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

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

{% block content %}
<h2>Bem-vindo ao Meu App!</h2>
<p>Esta é a página inicial.</p>
{% endblock %}
```

Código do templates/about.html:

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

{% block title %}About{% endblock %}

{% block content %}
<h2>Sobre Nós</h2>
<p>Esta é a página sobre.</p>
{% endblock %}
```

21:00 - 22:00: Exercício prático: Criando uma página web com Flask
Exercício Prático: Criando uma Página Web com Flask

Estrutura de Pastas:

```
meu_app/
│
├── app.py
├── templates/
│   ├── base.html
│   ├── home.html
│   └── about.html
├── static/
│   ├── css/
│   │   └── style.css
│   ├── js/
│   │   └── script.js
│   └── img/
│       └── logo.png
└── venv/
```

Configuração do app.py:

```
from flask import Flask, render_template

app = Flask(__name__)

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

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

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

Templates e Estilos:

Código do templates/base.html:

```
<!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 App{% endblock %}</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
    <header>
        <h1>Meu App</h1>
        <nav>
            <ul>
                <li><a href="{{ url_for('home') }}">Home</a></li>
                <li><a href="{{ url_for('about') }}">About</a></li>
            </ul>
        </nav>
    </header>
    <main>
        {% block content %}{% endblock %}
    </main>
    <script src="{{ url_for('static', filename='js/script.js') }}"></script>
</body>
</html>
```

Código do templates/home.html:

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

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

{% block content %}
<h2>Bem-vindo ao Meu App!</h2>
<p>Esta é a página inicial.</p>
{% endblock %}
```

Código do templates/about.html:

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

{% block title %}About{% endblock %}

{% block content %}
<h2>Sobre Nós</h2>
<p>Esta é a página sobre.</p>
{% endblock %}
```

Código do static/css/style.css:

```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    text-align: center;
    padding-top: 50px;
}
```

```
h1 {
    color: #333;
}

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

nav ul li {
    display: inline;
    margin: 0 10px;
}

nav ul li a {
    text-decoration: none;
    color: #333;
}

nav ul li a:hover {
    text-decoration: underline;
}
```

Código do static/js/script.js:

```
// Adicione seus scripts JavaScript aqui
console.log('Script carregado');
```

Executando a Aplicação

1. **No Terminal:** Navegue até a pasta do projeto e ative o ambiente virtual.
2. **Rodando a Aplicação:**
   ```
   python app.py
   ```
3. **Acessar no Navegador:** Abra o navegador e vá para http://127.0.0.1:5000/.

Dia 3: Estrutura das Aulas

18:30 - 19:00: Uso de templates no Flask

Uso de Templates:

Flask usa o motor de templates Jinja2 para renderizar HTML. Os templates são arquivos HTML que podem conter variáveis e expressões Python.

19:00 - 20:00: Passagem de variáveis para templates

Passagem de Variáveis para Templates:

No Flask, você pode passar variáveis para templates usando a função render_template.

20:00 - 21:00: Criação de layouts com templates

Criação de Layouts com Templates:

Você pode criar um layout base que outros templates herdam, facilitando a manutenção do design consistente em várias páginas.

21:00 - 22:00: Exercício prático: Criando uma página de layout com Flask
Exercício Prático: Criando uma Página de Layout com Flask

Estrutura de Pastas:

```
meu_app/
│
├── app.py
├── templates/
│   ├── base.html
│   ├── home.html
│   └── about.html
├── static/
│   ├── css/
│   │   └── style.css
│   ├── js/
│   │   └── script.js
│   └── img/
│       └── logo.png
└── venv/
```

Configuração do app.py:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    title = "Home"
    message = "Bem-vindo ao Meu App!"
    return render_template('home.html', title=title, message=message)

@app.route('/about')
def about():
    title = "About"
    message = "Esta é a página sobre."
    return render_template('about.html', title=title, message=message)

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

Templates e Estilos:

Código do templates/base.html:

```
<!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 App{% endblock %}</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
    <header>
        <h1>Meu App</h1>
        <nav>
            <ul>
                <li><a href="{{ url_for('home') }}">Home</a></li>
                <li><a href="{{ url_for('about') }}">About</a></li>
            </ul>
        </nav>
    </header>
    <main>
        {% block content %}{% endblock %}
    </main>
    <script src="{{ url_for('static', filename='js/script.js') }}"></script>
</body>
</html>
```

Código do templates/home.html:

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

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

{% block content %}
<h2>{{ message }}</h2>
<p>Esta é a página inicial.</p>
{% endblock %}
```

Código do templates/about.html:

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

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

{% block content %}
<h2>{{ message }}</h2>
<p>Esta é a página sobre.</p>
{% endblock %}
```

Código do static/css/style.css:

```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    text-align: center;
    padding-top: 50px;
}
```

```
h1 {
    color: #333;
}

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

nav ul li {
    display: inline;
    margin: 0 10px;
}

nav ul li a {
    text-decoration: none;
    color: #333;
}

nav ul li a:hover {
    text-decoration: underline;
}
```

Código do static/js/script.js:

```
// Adicione seus scripts JavaScript aqui
console.log('Script carregado');
```

Explicação e Passos

1. **Uso de Templates**: Usamos templates para separar a lógica de visualização do código Python. Os templates estão na pasta **templates**.
2. **Passagem de Variáveis**: No **app.py**, passamos variáveis para os templates usando a função **render_template**.
3. **Criação de Layouts**: Criamos um layout base (**base.html**) que define a estrutura comum para outras páginas. Outros templates (como **home.html** e **about.html**) herdam de **base.html** e preenchem os blocos definidos.

Executando a Aplicação

1. **No Terminal**: Navegue até a pasta do projeto e ative o ambiente virtual.
2. **Rodando a Aplicação**:
   `python app.py`
3. **Acessar no Navegador**: Abra o navegador e vá para **`http://127.0.0.1:5000/`**.

Dia 4: 08/08/2024

18:30 - 19:00: Introdução a formulários no Flask

Formulários são essenciais para interações de usuário em aplicações web. Flask facilita o uso de formulários através de extensões como Flask-WTF, que integra a biblioteca WTForms.

19:00 - 20:00: Biblioteca Flask-WTF para formulários

Flask-WTF simplifica a criação e validação de formulários no Flask, oferecendo uma integração com WTForms e suporte a CSRF (Cross-Site Request Forgery).

20:00 - 21:00: Validação de formulários

A validação de formulários garante que os dados recebidos do usuário estejam no formato correto e atendam aos requisitos definidos.

21:00 - 22:00: Exercício prático: Criando e validando um formulário com Flask-WTF

Exercício Prático: Criando e Validando um Formulário com Flask-WTF

Estrutura de Pastas:

```
meu_app/
│
├── app.py
├── forms.py
├── templates/
│   ├── base.html
│   ├── home.html
│   └── contact.html
├── static/
│   ├── css/
│   │   └── style.css
│   ├── js/
│   │   └── script.js
│   └── img/
│       └── logo.png
└── venv/
```

Instalação de Dependências:

No terminal, instale as dependências necessárias:

```
pip install Flask Flask-WTF
```

Configuração do app.py:

```
from flask import Flask, render_template, redirect, url_for, flash
from forms import ContactForm

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

```
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/contact', methods=['GET', 'POST'])
def contact():
    form = ContactForm()
    if form.validate_on_submit():
        flash('Formulário enviado com sucesso!', 'success')
        return redirect(url_for('home'))
    return render_template('contact.html', form=form)

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

Configuração do forms.py:

```
from flask_wtf import FlaskForm
from wtforms import StringField, TextAreaField, SubmitField
from wtforms.validators import DataRequired, Email

class ContactForm(FlaskForm):
    name = StringField('Nome', validators=[DataRequired()])
    email = StringField('Email', validators=[DataRequired(), Email()])
    message = TextAreaField('Mensagem', validators=[DataRequired()])
    submit = SubmitField('Enviar')
```

Templates e Estilos:

Código do templates/base.html:

```
<!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 App{% endblock %}</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
    <header>
        <h1>Meu App</h1>
        <nav>
            <ul>
                <li><a href="{{ url_for('home') }}">Home</a></li>
                <li><a href="{{ url_for('contact') }}">Contact</a></li>
            </ul>
        </nav>
    </header>
    <main>
        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                <ul class="flashes">
                {% for category, message in messages %}
                    <li class="{{ category }}">{{ message }}</li>
                {% endfor %}
                </ul>
            {% endif %}
        {% endwith %}
```

```
        {% block content %}{% endblock %}
    </main>
    <script src="{{ url_for('static', filename='js/script.js') }}"></script>
</body>
</html>
```

Código do templates/home.html:

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

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

{% block content %}
<h2>Bem-vindo ao Meu App!</h2>
<p>Esta é a página inicial.</p>
{% endblock %}
```

Código do templates/contact.html:

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

{% block title %}Contact{% endblock %}

{% block content %}
<h2>Contato</h2>
<form method="POST" action="">
    {{ form.hidden_tag() }}
    <div>
        {{ form.name.label }}<br>
        {{ form.name(size=32) }}<br>
        {% for error in form.name.errors %}
            <span class="error">{{ error }}</span>
        {% endfor %}
    </div>
    <div>
        {{ form.email.label }}<br>
        {{ form.email(size=32) }}<br>
        {% for error in form.email.errors %}
            <span class="error">{{ error }}</span>
        {% endfor %}
    </div>
    <div>
        {{ form.message.label }}<br>
        {{ form.message(rows=5) }}<br>
        {% for error in form.message.errors %}
            <span class="error">{{ error }}</span>
        {% endfor %}
    </div>
    <div>
        {{ form.submit() }}
    </div>
</form>
{% endblock %}
```

Código do static/css/style.css:

```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    text-align: center;
    padding-top: 50px;
}

h1 {
    color: #333;
}

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

nav ul li {
    display: inline;
    margin: 0 10px;
}

nav ul li a {
    text-decoration: none;
    color: #333;
}

nav ul li a:hover {
    text-decoration: underline;
}

.error {
    color: red;
}

.flashes {
    list-style-type: none;
    padding: 0;
    color: green;
}
```

Código do static/js/script.js:

```
// Adicione seus scripts JavaScript aqui
console.log('Script carregado');
```

Explicação e Passos

1. **Introdução a Formulários**: Introduzimos a criação de formulários HTML básicos.
2. **Biblioteca Flask-WTF**: Explicamos como usar Flask-WTF para facilitar a criação e validação de formulários.
3. **Validação de Formulários**: Mostramos como validar formulários usando os validadores do WTForms.
4. **Exercício Prático**: Criamos um formulário de contato e implementamos sua validação.

Executando a Aplicação

1. **No Terminal**: Navegue até a pasta do projeto e ative o ambiente virtual.
2. **Rodando a Aplicação**:
   `python app.py`
3. Acessar no Navegador: Abra o navegador e vá para `http://127.0.0.1:5000/`.