

## Ficha de Avaliação Final

**Curso:** UFCD 10793

**UFCD/Módulo/Temática:** UFCD 10793 - **Fundamentos de Python**

**Ação:** **10793\_2/AT & 10793\_5/N**

**Formador/a:** Sandra Liliana Meira de Oliveira

**Data:** março de 2025

**Nome do Formando/a:**

O presente documento é composto por uma parte teórica e uma parte prática. A resolução da parte prática corresponde à realização da Ficha de Avaliação Final.

|  |   |
|--|---|
| Introdução ao Flask .....  | 2 |
| Passo 1: Criar e configurar o ambiente .....                             | 5 |
| Passo 3: Criar a aplicação Flask.....                                    | 6 |
| Passo 4: Criar a base de dados e os modelos .....                        | 6 |
| Passo 5: Criar templates HTML com Bootstrap.....                         | 7 |
| Passo 6: Testar a aplicação .....  | 8 |
| Passo 7: Alterações à estrutura para evitar referências circulares ..... | 8 |

## Teoria

### Introdução ao Flask

O Flask é um framework em Python, amplamente reconhecido pela sua simplicidade e facilidade de utilização, constituindo-se como uma opção apelativa para programadores interessados em desenvolver aplicações web, evitando a complexidade inerente a outros frameworks mais abrangentes.

A simplicidade do Flask é resultante da sua natureza leve e minimalista. Muitas vezes é referido como um "micro-framework", uma vez que não impõe aos programadores regras ou estruturas rígidas e não possui funcionalidades excedentes, como camadas de abstracção de bases de dados ou um sistema ORM (Mapeamento Objecto-Relacional) pré-definido.

O Flask proporciona as ferramentas e bibliotecas essenciais que possibilitam aos programadores construir aplicações web de forma célere e eficiente. Um dos seus propósitos cruciais é manter as aplicações tão descomplicadas e livres de componentes desnecessários quanto possível, permitindo, simultaneamente, que os programadores adicionem as ferramentas que necessitam.

Concebido com a intenção de ser fácil de aprender e de implementar, o Flask permite que os programadores possam focar-se na lógica específica da aplicação que estão a criar, sem terem de se preocupar com pormenores relacionados com o protocolo HTTP ou com o manuseamento de pedidos e respostas web, algo que o Flask gera com mestria.

A estruturação de um projeto Flask, embora flexível, segue certas convenções e práticas recomendadas para assegurar que a aplicação seja escalável e de fácil manutenção. Aqui fica uma descrição da estrutura base típica de um projecto Flask:

Onde:

A estruturação de um projeto Flask, embora flexível, segue certas convenções e práticas

recomendadas para assegurar que a aplicação seja escalável e de fácil manutenção. Aqui fica uma descrição da estrutura base típica de um projecto Flask:

1. **/nome\_do\_projeto**: A pasta raiz que contém todo o projeto Flask.

1.1. **/venv**: Uma pasta que geralmente armazena o ambiente virtual do Python, mantendo as dependências do projeto isoladas.

1.2. **/app**: Esta pasta é o núcleo da aplicação Flask.

1.2.1. **/static**: Aqui são armazenados todos os ficheiros estáticos, como CSS, JavaScript, imagens, etc.

1.2.1.1. **/css**: Pasta destinada a ficheiros de estilos CSS.

1.2.1.2. **/js**: Pasta destinada a ficheiros JavaScript.

1.2.1.3. **/img**: Pasta destinada a imagens.

1.2.2. **/templates**: Contém os ficheiros HTML que serão renderizados pelo Flask. **index.html** é um exemplo comum de um ficheiro de template.

1.2.3. **\_\_init\_\_.py**: Este ficheiro torna a pasta **/app** num pacote Python e é onde a aplicação Flask é geralmente criada.

1.2.4. **routes.py**: Este ficheiro define as rotas da aplicação, ou seja, os diferentes endereços URL a que a aplicação irá responder e a lógica associada a cada um deles.

1.3. **config.py**: Utilizado para armazenar configurações que a aplicação precisa, como chaves secretas, configurações de base de dados, e outras variáveis de configuração.

1.4. **run.py**: Este é o ficheiro que é executado para iniciar a aplicação Flask. Geralmente cria uma instância da aplicação e executa-a.

Seguem-se alguns exemplos de código:



## Prática

Neste exercício os ficheiros route.py, init.py e run.py são parte integrante do ficheiro app.py. Logo para **correres** a aplicação dever correr o ficheiro **app.py**

Não temos ficheiros na pasta static porque estamos a usar a framework bootstrap que já inclui css e js.

Aplicação em Flask com base de dados SQLite passo a passo com posterior integração da framework bootstrap.

### Passo 1: Criar e configurar o ambiente

1. Criar uma pasta para o projeto

Abre o VS Code e cria uma nova pasta para o projeto, por exemplo, **flask\_app**.

2. Criar e ativar um ambiente virtual

Abre o terminal no VS Code e executa: **python -m venv venv**

3. Depois, ativa o ambiente virtual:

a. **Windows**: venv\Scripts\activate

b. **Linux/Mac**: source venv/bin/activate

4. **Instalar as dependências**

Instala o Flask e outras bibliotecas necessárias: **pip install flask flask-sqlalchemy flask-bootstrap**

### Passo 2: Criar a estrutura do projeto

Dentro da pasta **flask\_app**, cria a seguinte estrutura:

```
flask_app/  
| — templates/ # HTML templates
```

```
|—— static/      # CSS, JS, Bootstrap
|—— app.py       # Código principal do Flask
|—— models.py    # Modelos da base de dados
|—— forms.py     # Formulários Flask-WTF (opcional)
|—— database.db   # Base de dados SQLite (criada automaticamente)
|—— venv/        # Ambiente virtual (não precisa mexer aqui)
```

## Passo 3: Criar a aplicação Flask

No ficheiro **app.py**, adiciona o código base:

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

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

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

## Passo 4: Criar a base de dados e os modelos

No ficheiro **models.py**, cria uma tabela de exemplo:

```
from app import db

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)

    def __repr__(self):
```

```
return f'<User {self.name}>'
```

Cria o ficheiro **create\_db.py** na pasta flak\_app com o seguinte código. Executa o mesmo de seguida. A base de dados database.db será criada.

## Passo 5: Criar templates HTML com Bootstrap

Na pasta templates/ cria o ficheiro **layout.html** para servir como template base:

```
<!DOCTYPE html>
<html lang="pt">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flask App</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
    <nav class="navbar navbar-dark bg-dark">
        <div class="container">
            <a class="navbar-brand" href="/">Minha Flask App</a>
        </div>
    </nav>

    <div class="container mt-4">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

Agora, cria **index.html** dentro de templates/:

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

{% block content %}
    <h1 class="text-center">Bem-vindo à Minha Aplicação Flask!</h1>
{% endblock %}
```

## Passo 6: Testar a aplicação

Corre o servidor Flask. Para tal executa a script **app.py** e abre o url <http://127.0.0.1:5000> no browser.

## Passo 7: Alterações à estrutura para evitar referências circulares

Cria um novo ficheiro chamado **extensions.py** para separar a db com o seguinte código:

```
# extensions.py
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
```

altera o ficheiro **app.py** para:

```
# app.py
from flask import Flask
from extensions import db
from models import User # Isto agora já funciona sem ciclo

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db.init_app(app)
```

altera o ficheiro **models.py** para:

```
from extensions import db

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)

    def __repr__(self):
        return f'User {self.name}'
```

altera o ficheiro **app.py** para:

```
# app.py
from flask import Flask, request, redirect, url_for, render_template
from extensions import db
```

```
from models import User # Isto agora já funciona sem ciclo

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db.init_app(app)

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

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

altera o ficheiro **app.py** para podermos adicionar um utilizador acrescentando nova rota:

```
# app.py
from flask import Flask, request, redirect, url_for, render_template
from extensions import db
from models import User # Isto agora já funciona sem ciclo

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db.init_app(app)

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

@app.route('/add_user', methods=['GET', 'POST'])
def add_user():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        new_user = User(name=name, email=email)
        db.session.add(new_user)
        db.session.commit()
        return redirect(url_for('home'))

    return render_template('add_user.html')

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

No ficheiro **templates/add\_user.html**, cria o formulário:

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

{% block content %}
    <h2 class="text-center">Adicionar Usuário</h2>
    <form method="POST" class="container">
        <div class="mb-3">
            <label class="form-label">Nome</label>
            <input type="text" class="form-control" name="name" required>
        </div>
        <div class="mb-3">
            <label class="form-label">Email</label>
            <input type="email" class="form-control" name="email" required>
        </div>
        <button type="submit" class="btn btn-primary">Adicionar</button>
    </form>
{% endblock %}
```

No ficheiro **layout.html**, adiciona um botão no menu para a página de adicionar utilizador:

```
<a class="btn btn-primary" href="{{ url_for('add_user') }}>Adicionar Utilizador</a>
```

Vamos agora mostrar os utilizadores registados.

No ficheiro **app.py** faz as seguintes alterações:

```
# app.py
from flask import Flask, request, redirect, url_for, render_template
from extensions import db
from models import User # Isto agora já funciona sem ciclo

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db.init_app(app)

@app.route('/')
def home():
    users = User.query.all()
    return render_template('index.html', users=users)

@app.route('/add_user', methods=['GET', 'POST'])
def add_user():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        new_user = User(name=name, email=email)
        db.session.add(new_user)
```

```
    db.session.commit()
    return redirect(url_for('home'))

return render_template('add_user.html')

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

Altera o template **index.html** para exibir a lista de utilizadores

## 👉 Conclusão

Neste momento temos uma aplicação Flask funcional com:

- Base de dados SQLite
- Integração com Bootstrap
- registo e listagem de utilizadores
- Estrutura modular usando templates

Vamos expandir a aplicação Flask para incluir:

- Adicionar, Editar e Apagar Utilizadores
- Registo de Picagens (Entradas e Saídas)

No ficheiro **models.py** adiciona um modelo para armazenar as picagens

```
from datetime import datetime
from extensions import db

from datetime import datetime
from extensions import db

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)
    picagens = db.relationship('Picagem', backref='user', lazy=True, cascade="all, delete")

    def __repr__(self):
        return f'User {self.name}'
```

```
class Picagem(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    timestamp = db.Column(db.DateTime, default=datetime.utcnow)
    tipo = db.Column(db.String(10), nullable=False) # "Entrada" ou "Saída"

    def __repr__(self):
        return f'<Picagem {self.tipo} - {self.timestamp}>'
```

Apaga o ficheiro database.db

Agora, atualiza a base de dados correndo o script **create\_db.py**

Vamos agora acrescentar novas rotas no ficheiro **app.py** para:

- Listar utilizadores com opções de editar e apagar
- Adicionar Utilizador
- Editar Utilizador
- Apagar Utilizador

O ficheiro **app.py** deverá ficar igual a:

```
# app.py
from flask import Flask, request, redirect, url_for, render_template
from extensions import db
from models import User # Isto agora já funciona sem ciclo

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db.init_app(app)

@app.route('/')
def home():
    users = User.query.all()
    return render_template('index.html', users=users)

@app.route('/add_user', methods=['GET', 'POST'])
def add_user():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        new_user = User(name=name, email=email)
        db.session.add(new_user)
        db.session.commit()
        return redirect(url_for('home'))
    return render_template('add_user.html')
```

```
@app.route('/edit_user/<int:user_id>', methods=['GET', 'POST'])
def edit_user(user_id):
    user = User.query.get_or_404(user_id)
    if request.method == 'POST':
        user.name = request.form['name']
        user.email = request.form['email']
        db.session.commit()
        return redirect(url_for('home'))
    return render_template('edit_user.html', user=user)

@app.route('/delete_user/<int:user_id>', methods=['POST'])
def delete_user(user_id):
    user = User.query.get_or_404(user_id)
    db.session.delete(user)
    db.session.commit()
    return redirect(url_for('home'))

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

Modifica agora o ficheiro **index.html** para listar os utilizadores com botões de editar/apagar.

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

{% block content %}
    <h1 class="text-center">Usuários</h1>
    <a class="btn btn-success mb-3" href="{{ url_for('add_user') }}">Adicionar Utilizador</a>

    <table class="table">
        <thead>
            <tr>
                <th>ID</th>
                <th>Nome</th>
                <th>Email</th>
                <th>Ações</th>
            </tr>
        </thead>
        <tbody>
            {% for user in users %}
            <tr>
                <td>{{ user.id }}</td>
                <td>{{ user.name }}</td>
                <td>{{ user.email }}</td>
                <td>
                    <a href="{{ url_for('edit_user', user_id=user.id) }}" class="btn btn-primary">Editar</a>
                    <a href="{{ url_for('delete_user', user_id=user.id) }}" class="btn btn-danger">Apagar</a>
                </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
{% endblock %}
```

```
<form action="{{ url_for('delete_user', user_id=user.id) }}" method="POST" style="display:inline;">
    <button type="submit" class="btn btn-danger">Apagar</button>
</form>
</td>
</tr>
{% endfor %}
</tbody>
</table>
{% endblock %}
```

Cria agora o template **edit\_user.html** com o seguinte código:

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

{% block content %}
    <h2 class="text-center">Editar Utilizador</h2>
    <form method="POST" class="container">
        <div class="mb-3">
            <label class="form-label">Nome</label>
            <input type="text" class="form-control" name="name" value="{{ user.name }}" required>
        </div>
        <div class="mb-3">
            <label class="form-label">Email</label>
            <input type="email" class="form-control" name="email" value="{{ user.email }}" required>
        </div>
        <button type="submit" class="btn btn-primary">Atualizar</button>
    </form>
{% endblock %}
```

Altera o ficheiro **layout.html** para (remover botão inserir utilizador previamente criado):

```
<!DOCTYPE html>
<html lang="pt">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flask App</title>
    <link rel="stylesheet"
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
```

```
<div class="container mt-4">
    {% block content %}{% endblock %}
</div>
</body>
</html>
```

Vamos agora implementar o controlo de picagens:

Comecemos por criar a rota para registrar picagens no ficheiro **app.py**:

O código do ficheiro deverá ficar igual a:

```
# app.py
from flask import Flask, request, redirect, url_for, render_template
from extensions import db
from models import User, Picagem # Isto agora já funciona sem ciclo
from datetime import datetime

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db.init_app(app)

@app.route('/')
def home():
    users = User.query.all()
    return render_template('index.html', users=users)

@app.route('/add_user', methods=['GET', 'POST'])
def add_user():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        new_user = User(name=name, email=email)
        db.session.add(new_user)
        db.session.commit()
        return redirect(url_for('home'))
    return render_template('add_user.html')

@app.route('/edit_user/<int:user_id>', methods=['GET', 'POST'])
def edit_user(user_id):
    user = User.query.get_or_404(user_id)
    if request.method == 'POST':
        user.name = request.form['name']
        user.email = request.form['email']
        db.session.commit()
        return redirect(url_for('home'))
```

```

    return render_template('edit_user.html', user=user)

@app.route('/delete_user<int:user_id>', methods=['POST'])
def delete_user(user_id):
    user = User.query.get_or_404(user_id)
    db.session.delete(user)
    db.session.commit()
    return redirect(url_for('home'))

from datetime import datetime

@app.route('/picagem<int:user_id><tipo>', methods=['POST'])
def registrar_picagem(user_id, tipo):
    if tipo not in ["Entrada", "Saída"]:
        return "Tipo inválido", 400

    user = User.query.get_or_404(user_id)
    nova_picagem = Picagem(user_id=user.id, tipo=tipo)
    db.session.add(nova_picagem)
    db.session.commit()
    return redirect(url_for('home'))


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

```

Vamps agora atualizar o template **index.html** para adicionar botões de picagem:

O código deverá ficar como:

```

{% extends "layout.html" %}

{% block content %}
    <h1 class="text-center">Utilizadores</h1>
    <a class="btn btn-success mb-3" href="{{ url_for('add_user') }}>Adicionar
Utilizador</a>

    <table class="table">
        <thead>
            <tr>
                <th>ID</th>
                <th>Nome</th>
                <th>Email</th>
                <th>Ações</th>
            </tr>
        </thead>

```

```

<tbody>
    {% for user in users %}
    <tr>
        <td>{{ user.id }}</td>
        <td>{{ user.name }}</td>
        <td>{{ user.email }}</td>

        <td>
            <a href="{{ url_for('edit_user', user_id=user.id) }}">
                Editar
            </a>
            <form action="{{ url_for('delete_user', user_id=user.id) }}"
                method="POST" style="display:inline;">
                <button type="submit" class="btn btn-danger">Apagar</button>
            </form>
            <form action="{{ url_for('registar_picagem', user_id=user.id,
                tipo='Entrada') }}" method="POST" style="display:inline;">
                <button type="submit" class="btn btn-success">Entrada</button>
            </form>
            <form action="{{ url_for('registar_picagem', user_id=user.id,
                tipo='Saída') }}" method="POST" style="display:inline;">
                <button type="submit" class="btn btn-warning">Saída</button>
            </form>
        </td>
    </tr>
    {% endfor %}
</tbody>
</table>
{% endblock %}

```

Vamos agora criar a página para mostrar picagens.

Para tal começemos por acrescentar uma nova rota ao ficheiro **app.py**

**O ficheiro referido deverá ficar igual a:**

```

# app.py
from flask import Flask, request, redirect, url_for, render_template
from extensions import db
from models import User, Picagem # Isto agora já funciona sem ciclo
from datetime import datetime

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'

```

```
db.init_app(app)

@app.route('/')
def home():
    users = User.query.all()
    return render_template('index.html', users=users)

@app.route('/add_user', methods=['GET', 'POST'])
def add_user():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        new_user = User(name=name, email=email)
        db.session.add(new_user)
        db.session.commit()
        return redirect(url_for('home'))
    return render_template('add_user.html')

@app.route('/edit_user/<int:user_id>', methods=['GET', 'POST'])
def edit_user(user_id):
    user = User.query.get_or_404(user_id)
    if request.method == 'POST':
        user.name = request.form['name']
        user.email = request.form['email']
        db.session.commit()
        return redirect(url_for('home'))
    return render_template('edit_user.html', user=user)

@app.route('/delete_user/<int:user_id>', methods=['POST'])
def delete_user(user_id):
    user = User.query.get_or_404(user_id)
    db.session.delete(user)
    db.session.commit()
    return redirect(url_for('home'))

from datetime import datetime

@app.route('/picagem/<int:user_id>/<tipo>', methods=['POST'])
def registrar_picagem(user_id, tipo):
    if tipo not in ["Entrada", "Saída"]:
        return "Tipo inválido", 400

    user = User.query.get_or_404(user_id)
    nova_picagem = Picagem(user_id=user.id, tipo=tipo)
    db.session.add(nova_picagem)
```

```
db.session.commit()
return redirect(url_for('home'))

@app.route('/picagens')
def listar_picagens():
    picagens = Picagem.query.order_by(Picagem.timestamp.desc()).all()
    return render_template('picagens.html', picagens=picagens)

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

Cria o template **picagens.html** com o seguinte código:

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

{% block content %}
    <h1 class="text-center">Registo de Picagens</h1>
    <table class="table">
        <thead>
            <tr>
                <th>Nome</th>
                <th>Tipo</th>
                <th>Data e Hora</th>
            </tr>
        </thead>
        <tbody>
            {% for picagem in picagens %}
            <tr>
                <td>{{ picagem.user.name }}</td>
                <td>{{ picagem.tipo }}</td>
                <td>{{ picagem.timestamp }}</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
{% endblock %}
```

No ficheiro **layout.html** vamos acrescentar Alguns urls para mostrar as picagens e facilitar o acesso a outras ações. O template deverá ficar como:

```
<!DOCTYPE html>
<html lang="pt">
<head>
    <meta charset="UTF-8">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Gestão de Picagens</title>
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>

    <!-- Barra de navegação -->
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container">
            <a class="navbar-brand" href="{{ url_for('home') }}>Minha Flask
App</a>
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarNav">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav me-auto">
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url_for('home') }}>Início</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url_for('add_user') }}>Adicionar Utilizador</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url_for('listar_picagens') }}>Picagens</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>

    <!-- Conteúdo principal -->
    <div class="container mt-4">
        {% block content %}{% endblock %}
    </div>

    <!-- Scripts do Bootstrap (para toggle em mobile funcionar) -->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">
    </script>
</body>
</html>
```

