Criando uma agenda com Python, Flask e SQLite

O projeto elaborado nesta atividade é uma agenda (CRUD), onde realizaremos a estruturação de pastas e conteúdos passo a passo, do essencial até um programa com algumas validações/formatações das saídas de dados. Utilizaremos o GIT e demais aprendizados anteriores das aulas de Python no desenvolvimento da solução.

1. Abra um console/terminal (CMD, PowerShell, Bash ou Zsh) e entre em seu local de desenvolvimento de projetos.

2. Crie uma pasta “agenda” para elaboração deste projeto

```
$ mkdir agenda
$ cd agenda
agenda$ _
```

3. Inicialize seu diretório com o git para controle de versionamento

```
$ git init
```

4. Abra o Visual Studio Code para edição desta atividade. A partir deste momento, utilizaremos o VSCode para codificação e utilização de terminal de comandos embutido nesta interface para elaborar esta atividade.

5. Criaremos um ambiente virtual (venv, virtual environment) no python para instalação e utilização dos recursos necessários neste projeto

```
$ python3 -m venv venv
-ou-
> python -m venv venv
```

A sintaxe deste comando é utilizar o python para criar (-m) o venv com o nome_do_ambiente_virtual, que neste caso também foi chamado de venv. Porém ainda não estamos utilizando o ambiente virtual que acabamos de criar, para isso teremos que executar o arquivo activate que esta no caminho venv/bin/activate. Note que o sentido das barras pode ser diferente entre Linux e Windows

```
$ source venv/bin/activate
-ou-
> .venv\Scripts\activate
```

6. Feito isto, vamos instalar as dependências necessárias para o projeto

```
$ pip3 install Flask Flask-SQLAlchemy
-ou-
> pip install Flask Flask-SQLAlchemy
```

A primeira parte do projeto está pronta. Vamos agora entender melhor como estruturar os arquivos que utilizaremos para criar este CRUD.

A estrutura inicial do projeto será esta:

```
/agenda
  ├── app.py           <- Programa principal
  ├── instance/        <- Pasta com o banco de dados
  │   └── database.db  <- Arquivo do banco SQLite com tabela agenda
  └── templates/       <- Pasta com o conteúdo HTML
      ├── base.html    <- Página principal do programa
      ├── list.html    <- Página para listar contatos da agenda
      ├── add.html     <- Página para adicionar novo contato na agenda
      └── edit.html    <- Página para editar conteúdo da agenda
```

Considerando o que já foi abordado nas aulas anteriores, teremos o seguinte conteúdo:

app.py

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

class Contact(db.Model):
    __tablename__ = 'agenda'  # Nome da tabela
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    phone = db.Column(db.String(15), nullable=False)
    email = db.Column(db.String(100), nullable=False)

@app.route('/')
def index():
    contacts = Contact.query.all()
    return render_template('list.html', contacts=contacts)

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

@app.route('/edit/<int:id>', methods=['GET', 'POST'])
def edit(id):
    contact = Contact.query.get_or_404(id)
    if request.method == 'POST':
        contact.name = request.form['name']
        contact.phone = request.form['phone']
        contact.email = request.form['email']
        db.session.commit()
        return redirect(url_for('index'))
    return render_template('edit.html', contact=contact)

@app.route('/delete/<int:id>')
def delete(id):
    contact = Contact.query.get_or_404(id)
    db.session.delete(contact)
    db.session.commit()
    return redirect(url_for('index'))

# Verifica se o banco de dados existe e cria a tabela se necessário
```

```
def create_db():
    if not os.path.exists('database.db'):
        with app.app_context():
            db.create_all()  # Cria a(s) tabela(s)
create_db() # Chama a função para verificar e criar o banco de dados

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

Página principal

templates/base.html

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <title>Agenda</title>
</head>
<body>
    <header>
        <h1>Agenda</h1>
        <nav>
            <a href="/">Listar</a>
            <a href="/add">Inserir</a>
        </nav>
    </header>
    <main>
        {% block content %}{% endblock %}
    </main>
</body>
</html>
```

Página para adicionar os contatos

templates/add.html

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

{% block content %}
    <h2>Adicionar Contato</h2>
    <form method="POST">
        <label>Nome Completo:</label>
        <input type="text" name="name" required>
        <label>Número de Telefone:</label>
        <input type="text" name="phone" required>
        <label>E-mail:</label>
        <input type="email" name="email" required>
        <button type="submit">Adicionar</button>
    </form>
{% endblock %}
```

Página para listar os contatos

templates/list.html

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

{% block content %}
    <h2>Contatos</h2>
    <ul>
        {% for contact in contacts %}
            <li>
                {{ contact.name }} - {{ contact.phone }} - {{ contact.email }}
                <a href="/edit/{{ contact.id }}">Editar</a>
                <a href="/delete/{{ contact.id }}">Excluir</a>
            </li>
        {% endfor %}
    </ul>
{% endblock %}
```

Página para editar os contatos

templates/edit.html

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

{% block content %}
    <h2>Editar Contato</h2>
    <form method="POST">
        <label>Nome Completo:</label>
        <input type="text" name="name" value="{{ contact.name }}" required>
        <label>Número de Telefone:</label>
        <input type="text" name="phone" value="{{ contact.phone }}" required>
        <label>E-mail:</label>
        <input type="email" name="email" value="{{ contact.email }}" required>
        <button type="submit">Salvar</button>
    </form>
{% endblock %}
```

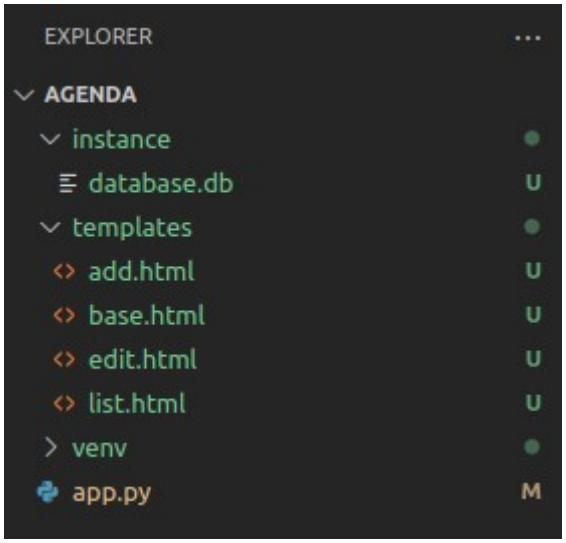
Para executar este projeto, basta utilizar:

```
$ python3 app.py
```

-ou-

```
> python app.py
```

Seu projeto estará assim em seu VSCode:



Verificando no terminal a situação (status) do seu versionamento de arquivos temos isto:

```
projetos/agenda$ git status

No ramo master

No commits yet

Arquivos não monitorados:
  (utilize "git add <arquivo>..." para incluir o que será submetido)
        app.py
        database.db
        templates/

nada adicionado ao envio mas arquivos não registrados estão presentes (use "git
add" to registrar)
```

Para o controle de versão, podemos adicionar arquivo por arquivo com uma descrição adequada para cada qual, onde colocamos detalhes de cada tarefa. Por exemplo:

Adiciono o arquivo “app.py” e verifico o status

```
/projetos/agenda$ git add app.py
/projetos/agenda$ git status
No ramo master

No commits yet

Mudanças a serem submetidas:
  (utilize "git rm --cached <arquivo>..." para não apresentar)
        new file:   app.py

Arquivos não monitorados:
  (utilize "git add <arquivo>..." para incluir o que será submetido)
        database.db
        templates/
```

Faço então o “`git commit`”

```
feat: principal app file

This file imports required libs, defines the database that will be
used; create all the routes required for CRUD and create the database
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# No ramo master
#
# Submissão inicial.
#
# Mudanças a serem submetidas:
#       new file:   app.py
#
# Arquivos não monitorados:
#       database.db
#       templates/
#
```

Observe que na primeira linha será o título do versionamento, seguido por uma descrição. Note que abaixo o texto em verde é um comentário que não aparecerá no histórico quando for enviado para seu github. Após salvar a mensagem do commit, o prompt mostrará algo como isto:

```
/projetos/agenda$ git commit
[master (root-commit) e964db2] feat: principal app file
 1 file changed, 53 insertions(+)
 create mode 100644 app.py
```

Caso seja verificado o status novamente, serão mostradas apenas as modificações que ainda não foram adicionadas para envio ao repositório no github.

Para seguir o fluxo completo de uso do github, verifique a documentação disponibilizada como materiais complementares da disciplina.

Exercícios:

1. Para ter um código mais legível, é conveniente desacoplar as rotas do programa principal. Faça isto colocando-as em um arquivo “routes.py”.

2. Crie uma estilização para sua agenda, criando um cabeçalho para a página e destacando os contatos da agenda.

3. Caso o banco de dados esteja vazio ao iniciar o programa, mostre uma mensagem informando isto e sugerindo a inserção de contatos.

4. Aplique uma formatação para os números de telefone mostrados no contato.

5. Faça um tratamento da string do nome para que o primeiro caractere de cada nome fique em letra maiúscula e o restante em minúsculas. Considere a expressão ‘de’, ‘da’, ‘do’ como termos que permanecem com todas as letras minúsculas.