Desenvolvimento de APIs e Microsserviços

Aula

Caio Eduardo do Prado Ireno
caio.ireno@faculdadeimpacta.com.br

requirements.txt

Criar o requirements.txt

O requirements.txt é um arquivo de texto usado para gerenciar as dependências de um projeto Python. Ele lista todos os pacotes Python necessários para executar o projeto, incluindo suas versões específicas, para que o ambiente de desenvolvimento possa ser facilmente replicado.

config.py

Criar o config.py

O arquivo **config.py** em uma aplicação Flask é crucial para centralizar e gerenciar as configurações do aplicativo, como chaves secretas, URLs de banco de dados e configurações de ambiente.

Ele facilita a customização, separação por ambiente e manutenção do aplicativo, seguindo um padrão de design reconhecido. Em resumo, o config.py simplifica o desenvolvimento e a gestão de configurações em aplicações Flask.

app.py

Criar o app.py

O arquivo app.py geralmente serve como ponto de entrada principal para o aplicativo Flask. Ele é responsável por integrar diferentes partes do aplicativo, como rotas, views e configurações, e iniciar o servidor Flask.

app.py

Integração de Componentes: O arquivo app.py importa e integra diferentes partes do aplicativo, como blueprints, rotas e configurações. Ele conecta esses componentes para construir o aplicativo Flask completo.

Registro de Blueprints: Os blueprints são usados para organizar e modularizar rotas relacionadas em um aplicativo Flask. O arquivo app.py geralmente registra os blueprints para tornar suas rotas disponíveis para o aplicativo principal.

app.py

Configurações do Aplicativo: O arquivo app.py pode importar configurações do arquivo config.py e aplicá-las ao aplicativo Flask. Isso permite que as configurações do aplicativo sejam centralizadas e facilmente gerenciadas em um único arquivo.

Inicialização do Servidor: O arquivo app.py geralmente contém um bloco if __name__ == '__main__' que inicia o servidor Flask quando o arquivo é executado diretamente. Isso permite que o aplicativo seja executado localmente durante o desenvolvimento.

Blueprint

Modular Applications with Blueprints — Flask Documentation (3.0.x) (palletsprojects.com)

Utilizando blueprint

Separar nossas rotas em vários locais.

Um blueprint em Flask é uma maneira de organizar e agrupar rotas relacionadas em um aplicativo Flask. Ele permite dividir uma aplicação em componentes menores e mais gerenciáveis, o que facilita a modularização e a escalabilidade do código.

Utilizando blueprint

Ao definir um blueprint, você pode agrupar rotas e views relacionadas em um arquivo ou pacote separado. Isso ajuda na organização do código, especialmente em aplicativos maiores, onde a estrutura do código pode se tornar complexa.

Utilizando blueprint

Principais características e benefícios dos blueprints em Flask:

Modularidade: Os blueprints permitem dividir uma aplicação em módulos independentes, cada um com suas próprias rotas, views e templates.

Reutilização: Os blueprints podem ser facilmente reutilizados em diferentes partes de uma aplicação ou em diferentes aplicativos Flask.

Utilizando blueprint

Manutenção: A separação de funcionalidades em blueprints facilita a manutenção do código, pois torna mais fácil localizar e atualizar partes específicas da aplicação.

Organização: Os blueprints ajudam a organizar o código de forma mais lógica e estruturada, o que torna o desenvolvimento e a manutenção do aplicativo mais eficientes.

Definição no nosso projeto

Para definir um blueprint em Flask, você geralmente cria uma instância de Blueprint e, em seguida, define rotas e views dentro desse blueprint.

Em seguida, você registra o blueprint no aplicativo principal usando o método app.register_blueprint(). Isso torna as rotas e views do blueprint disponíveis para o aplicativo principal.

Definição no nosso projeto

No nosso exemplo, vamos definir o blueprint chamado posts, no arquivo **posts/index.py** e registra-lo no arquivo **app.py.**

Isso permite que as rotas relacionadas aos posts sejam agrupadas e acessíveis como parte do aplicativo Flask principal.

Config

import os: Importa o módulo os, que fornece funcionalidades para interagir com o sistema operacional, como manipulação de diretórios e variáveis de ambiente.

```
import os
from flask import Flask
```

```
app = Flask(__name__)
app.config['HOST'] = '0.0.0.0'
app.config['PORT']=8000
app.config['DEBUG'] = True
```

DEBUG: Isso ativa o modo de depuração do Flask, que fornece informações detalhadas sobre erros e exceções durante o desenvolvimento.

HOST: configura o host para o aplicativo ser acessível em todas as interfaces de rede no servidor. Ou seja, o aplicativo será acessível de qualquer IP disponível no servidor.

PORT: configura a porta na qual o servidor Flask irá escutar por solicitações HTTP. Neste caso, o aplicativo será executado na porta 8000.

Blueprint

```
from flask import Blueprint

posts = Blueprint('posts',__name__)

@posts.route('/', methods=["GET"])
def main():
  return 'Posts routes'
```

Blueprint

posts = Blueprint('posts', __name__): Cria uma instância de blueprint chamada posts. O primeiro argumento é o nome do blueprint, e o segundo argumento (__name__) é o nome do módulo atual. Isso é usado pelo Flask para determinar a localização dos recursos estáticos, templates, entre outros.

@posts.route('/', methods=["GET"]): Define uma rota dentro do blueprint posts. Neste caso, a rota é '/', que é o caminho raiz. O argumento methods especifica os métodos HTTP permitidos para esta rota, neste caso, apenas o método GET.

Blueprint

def main(): Define uma função chamada main, que será chamada quando a rota '/' for acessada.

return 'Posts routes': Retorna a string 'Posts routes' como resposta à solicitação HTTP. Esta é a resposta que o cliente receberá quando acessar a rota definida.

Ou seja, esse trecho de código define um blueprint chamado posts, que contém uma rota raiz ('/') que responde apenas a solicitações GET com a string 'Posts routes'. Esse blueprint pode então ser registrado em um aplicativo Flask principal para que suas rotas estejam acessíveis no aplicativo.

app

```
import os
from config import app
from posts.index import posts

app.register_blueprint(posts)

if __name__ == '__main__':
  app.run(host=app.config["HOST"], port =
app.config['PORT'],debug=app.config['DEBUG'] )
```

app

from config import app: Importa a instância do aplicativo Flask (app) do módulo **config**. Presumivelmente, o arquivo **config.py** contém a instância do aplicativo Flask chamada app.

from posts.index import posts: Importa o **blueprint** chamado **posts** do módulo **posts.index**. O blueprint posts contém as rotas relacionadas a posts, como definido no arquivo **index.py** dentro do pacote posts.

app.register_blueprint(posts): Registra o blueprint posts no aplicativo Flask (app). Isso torna as rotas definidas no blueprint posts disponíveis para o aplicativo principal.

Atividade.

Vamos implementar nosso CRUD alunos/professores, nessa nova estrutura de projetos.

Obrigado!