Introdução ao Desenvolvimento Web com Python

O Que É Desenvolvimento Web?

- O desenvolvimento web cria aplicações acessadas por navegadores ou APIs.

Componentes básicos:

- Cliente: Geralmente um navegador ou dispositivo que envia requisições.
- Servidor: Responde às requisições, processa dados e entrega os resultados.

Aplicações Web Dinâmicas:

- Além de páginas estáticas, o desenvolvimento web moderno foca em aplicações dinâmicas que interagem com o usuário, processam dados e fornecem experiências personalizadas.

- Exemplos incluem redes sociais, lojas online, sistemas de gerenciamento e aplicativos de produtividade.

APIs (Interfaces de Programação de Aplicações):

- As APIs permitem que diferentes sistemas de software se comuniquem e troquem dados.
- No desenvolvimento web, as APIs são cruciais para integrar serviços de terceiros, como mapas, pagamentos e autenticação.
- Python é muito utilizado na construção de API's, com o auxílio de frameworks como o Flask ou o DjangoRestFramework.

Front-end e Back-end:

- O desenvolvimento web é frequentemente dividido em front-end (a interface com o usuário) e back-end (a lógica do servidor).

- Python é predominantemente usado no back-end, mas também pode ser integrado com tecnologias front-end através de APIs.

Protocolo HTTP

- O HTTP (HyperText Transfer Protocol) é usado na comunicação cliente-servidor.
- Principais métodos HTTP:

  - GET: Obter informações.
  - POST: Enviar dados para o servidor.
  - PUT: Atualizar dados.
  - DELETE: Remover dados.

Códigos de Status HTTP:

- Além dos métodos, os códigos de status HTTP indicam o resultado de uma requisição.
- Exemplos: 200 (OK), 404 (Não Encontrado), 500 (Erro Interno do Servidor).

Cabeçalhos HTTP:

- Os cabeçalhos HTTP fornecem informações adicionais sobre a requisição e a resposta.
- Eles podem controlar o cache, a autenticação, o tipo de conteúdo e muito mais.

HTTPS (HTTP Seguro):

- O HTTPS é uma versão segura do HTTP que criptografa a comunicação entre o cliente e o servidor.

- É essencial para proteger dados sensíveis, como senhas e informações de cartão de crédito.

Noções de Arquitetura Web

- Monólito:
  - Todo o sistema está em uma única aplicação.
  - Vantagem: Simples para projetos pequenos.
  - Desvantagem: Difícil de escalar e manter com o crescimento do sistema.

- Microsserviços:
  - O sistema é dividido em serviços independentes.
  - Vantagem: Escalabilidade, fácil manutenção.
  - Desvantagem: Mais complexo de implementar.

Arquitetura Serverless:

- Uma abordagem moderna que permite executar código sem gerenciar servidores.
- Serviços como AWS Lambda e Google Cloud Functions permitem criar aplicações escaláveis e econômicas.

Arquitetura de Micro Frontends:

- Um conceito que extende a ideia de microsserviços para o front-end, onde partes da interface do usuário são desenvolvidas e implementadas de forma independente.

Containers e Orquestração:

- Tecnologias como Docker e Kubernetes facilitam a criação, implantação e gerenciamento de aplicações em containers.

- Isso garante a consistência e a escalabilidade das aplicações.

Python no Desenvolvimento Web

- Python oferece frameworks como:
  - Flask: Minimalista, ótimo para projetos pequenos e APIs.
  - Django: Completo, inclui ORM, autenticação, templates, etc.

Django:

- ORM (Mapeamento Objeto-Relacional): O ORM do Django permite interagir com bancos de dados usando código Python, sem escrever SQL diretamente.

- Django REST Framework: Uma poderosa ferramenta para criar APIs RESTful com Django.

- Django Channels: Permite o desenvolvimento de aplicações web em tempo real.
- Outros Frameworks e Ferramentas:Além de Flask e Django, existem outros frameworks como Pyramid e FastAPI.

- Ferramentas como SQLAlchemy (ORM) e Requests (requisições HTTP) são amplamente utilizadas no desenvolvimento web com Python.

Flask:

- Extensões Flask: O Flask pode ser estendido com bibliotecas para adicionar funcionalidades como autenticação, bancos de dados e APIs.

- Flask Restful: Biblioteca que auxilia na criação de API's com o Flask.

Exercícios Práticos

Primeiro Servidor Web com Flask

- `Flask` : Framework para criar aplicações web.
- `@app.route('/')` : Define uma rota para a página inicial.
- `app.run(debug=True)` : Inicia o servidor.

Exercício 1: Configurar um Servidor Flask

Enunciado: Configure um servidor web básico com Flask que exibe `"Olá, mundo!"` na página inicial.

Exercício 2: Criar Rotas Diferentes

Enunciado: Adicione uma rota `/sobre` que exibe `"Esta é uma aplicação Flask básica."`.

Exercício 3: HTTP com Métodos Diferentes

Enunciado: Configure uma rota `/dados` que aceite requisições **GET** e **POST** e exiba mensagens diferentes para cada método.

Exercício 1: Criação de Rotas

- **Enunciado:** Crie uma aplicação Flask com três rotas:
  - `/` : Exibe `"Página Inicial"`.
  - `/contato` : Exibe `"Contato: email@dominio.com"`.
  - `/sobre` : Exibe `"Sobre nós"`.

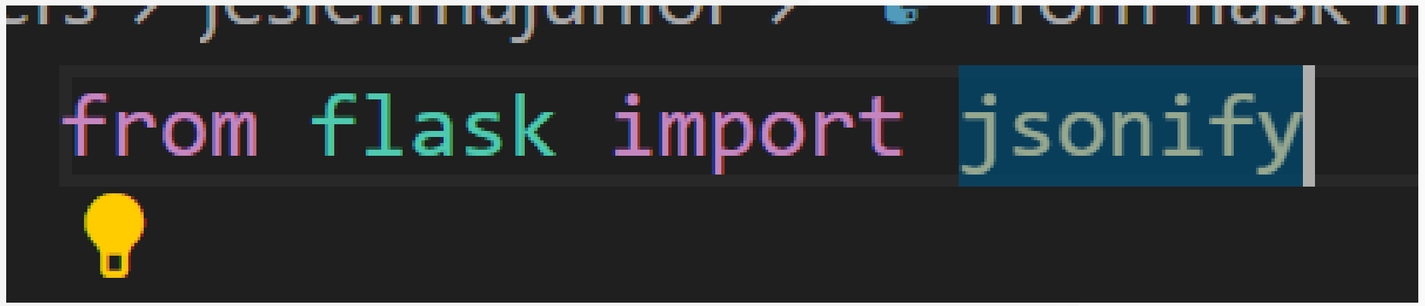
```
python
Copiar código
@app.route('/')
def home():
    return "Página Inicial"

@app.route('/contato')
def contato():
    return "Contato: email@dominio.com"

@app.route('/sobre')
def sobre():
    return "Sobre nós"
```

Exercício 2: Responder com JSON

Enunciado: Crie uma rota `/info` que retorne um JSON com as informações: `{"versao": "1.0", "autor": "Seu Nome"}`.



```
python
Copiar código
from flask import jsonify

@app.route('/info')
def info():
    return jsonify({"versao": "1.0", "autor": "Seu Nome"})
```

Exercício 3: Parâmetros na URL

Enunciado: Configure uma rota `/saudacao/<nome>` que receba um nome na URL e exiba `"Olá, <nome>!"`.

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

Exercício 4: Simular API com Métodos HTTP

Exercício 5: Aplicação com Template HTML

Enunciado: Crie uma rota `/html` que exiba uma página HTML simples com a mensagem `"Bem-vindo à minha aplicação!"`.