Programação Web com Python

Introdução ao Django

O que é um framework?

Um framework é uma estrutura de software predefinida que oferece um conjunto de ferramentas e componentes reutilizáveis para o desenvolvimento de aplicações. Ele fornece uma base sobre a qual os desenvolvedores podem construir sistemas mais complexos de forma mais eficiente e padronizada.

Imagine um framework como u ma caixa de ferramentas

O que é um django?

Django é um framework web de alto nível em Python

Criado para facilitar o desenvolvimento rápido e com design limpo e pragmático

django

Principais características:

django

1. **Rápido desenvolvimento.**
2. **DRY (Don't Repeat Yourself).**
3. **Alta segurança.**

Quando usar Django

- Aplicações web robustas e escaláveis.
- Projetos que necessitam de rapidez no desenvolvimento.

Aplicações desenvolvidas com django:

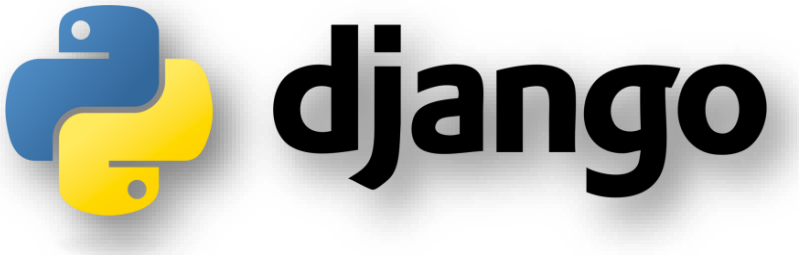
Aplicações desenvolvidas com django:

Motivo do Uso de Django: Django proporciona uma base sólida para lidar com o grande volume de dados e tráfego, oferecendo uma estrutura organizada e segura.

- **Motivo do Uso de Django**: Django facilita a construção de funcionalidades complexas e escaláveis, essenciais para um grande número de usuários ativos e conteúdo visual.

Motivo do Uso de Django: A flexibilidade e segurança do Django permitem a Mozilla desenvolver rapidamente soluções personalizadas e seguras para suas necessidades internas.

MVT no Django



MVT é a sigla para **Model-View-Template**, que é o padrão arquitetural usado pelo Django para organizar o desenvolvimento de aplicações web. Vamos entender cada um desses componentes de forma simples e didática:

Model (Modelo)

O **Model** é responsável por definir a estrutura dos dados e as regras de negócio da aplicação. Ele mapeia as tabelas do banco de dados para classes Python.

Exemplo Simples: Imagine que estamos criando um sistema de blog. Vamos definir um modelo para as postagens do blog.

python Copiar código

```python
# models.py
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

Aqui, o modelo Post tem três campos: title (título da postagem), content (conteúdo da postagem) e created_at (data e hora de criação da postagem).

View (Visão)

A **View** é responsável por lidar com a lógica da aplicação. Ela recebe as requisições dos usuários, processa os dados usando os modelos, e retorna uma resposta.

Exemplo Simples: Vamos criar uma view que exibe todas as postagens do blog

python Copiar código

```python
# views.py
from django.shortcuts import render
from .models import Post

def post_list(request):
    posts = Post.objects.all()
    return render(request, 'post_list.html', {'posts': posts})
```

Aqui, a view `post_list` consulta todas as postagens do banco de dados e as passa para um template chamado `post_list.html`.

Template (Modelo de Apresentação)

O **Template** é responsável por exibir os dados para os usuários. Ele define a estrutura e o design da página web.

Exemplo Simples: Vamos criar um template que exibe as postagens do blog.

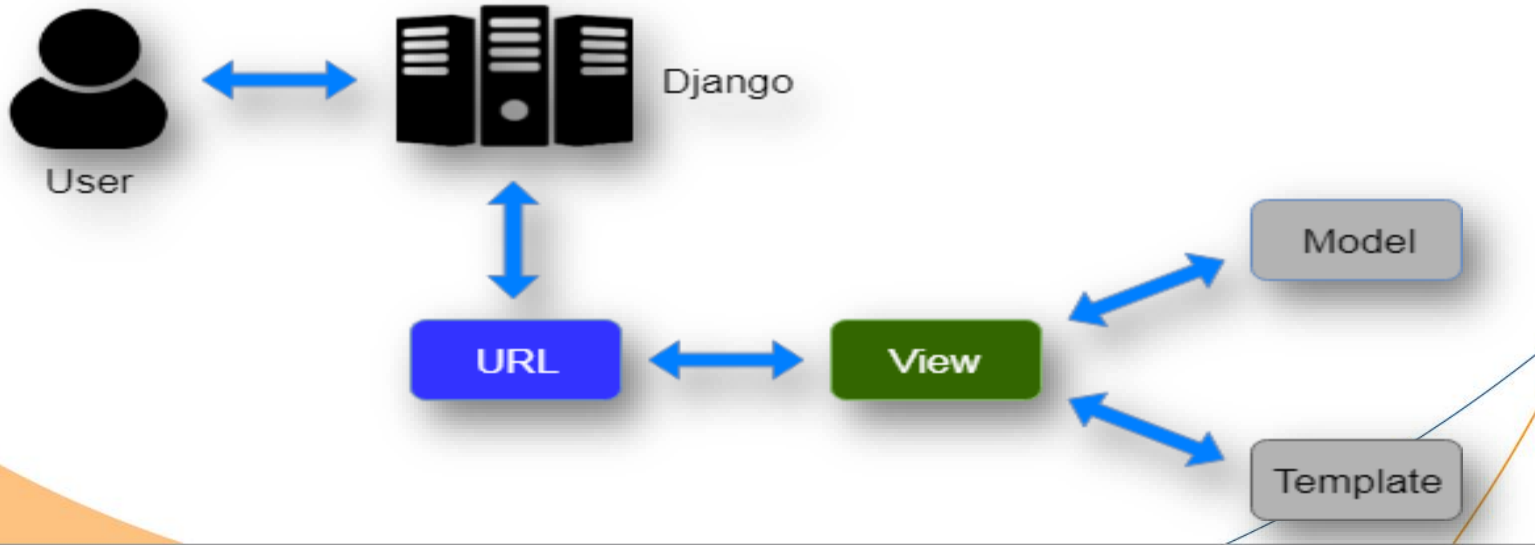
Aqui, o template post_list.html exibe o título, conteúdo e data de criação de cada postagem.

```
<!-- post_list.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Blog</title>
</head>
<body>
    <h1>Postagens do Blog</h1>
    {% for post in posts %}
        <h2>{{ post.title }}</h2>
        <p>{{ post.content }}</p>
        <p><em>Publicado em: {{ post.created_at }}</em></p>
    {% endfor %}
</body>
</html>
```

Template (Modelo de Apresentação)

Explicação do Fluxo:

- **Usuário**: O usuário acessa a URL para visualizar as postagens do blog.
- **Requisição HTTP**: A requisição é enviada ao servidor.
- **View (Visão)**: A view `post_list` recebe a requisição, consulta as postagens no modelo `Post` e passa os dados para o template `post_list.html`.
- **Template (Modelo de Apresentação)**: O template renderiza os dados e cria uma página HTML.
- **Resposta HTTP**: A página HTML é enviada de volta ao navegador do usuário.
- **Usuário**: O usuário vê a lista de postagens do blog no navegador.

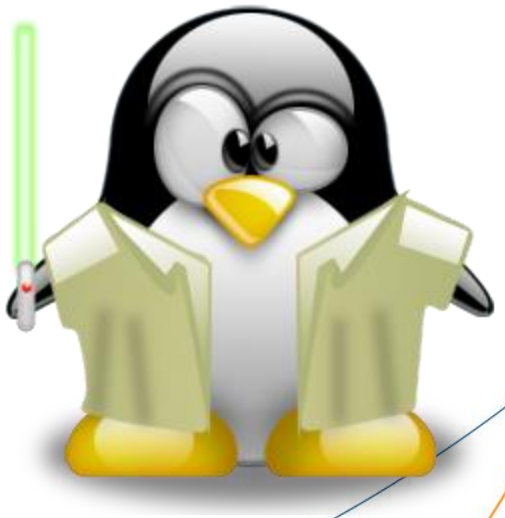


Conclusão

O padrão MVT do Django separa claramente as responsabilidades:

- **Model**: Define a estrutura e as regras dos dados.
- **View**: Contém a lógica de negócio e manipula as requisições.
- **Template**: Define como os dados são apresentados aos usuários.

Essa separação facilita o desenvolvimento, a manutenção e a escalabilidade da aplicação.



OBRIGADO!