

AULA 01

O QUE VEREMOS HOJE

01 DJANGO E CARACTERÍSTICAS

02 CONFIGURAÇÃO DE AMBIENTE

03 ESTRUTURA DO DJANGO

04 HTTP

05 URLS

06 VIEWS

O QUE É DJANGO?

Django é um framework web de código aberto, escrito em Python, que facilita o desenvolvimento rápido e seguro de aplicações web.

Ele oferece ferramentas prontas para ajudar desenvolvedores a construir desde sites simples até sistemas complexos de maneira mais eficiente.

[illegible]

CARACTERÍSTICAS DO DJANGO

Funcionalidades prontas

O Django vem com tudo o que você precisa para criar uma aplicação web, como autenticação, ORM, administração e segurança, facilitando ainda mais o desenvolvimento.

Padrão MVT (Model-View-Template)

É um padrão amplamente conhecido no desenvolvimento web, onde a aplicação é dividida em 3 componentes principais:

- **Model:** Estrutura e manipulação dos dados. Cuida da interação com o banco de dados, garantindo que os dados sejam armazenados e recuperados corretamente.
- **View:** Fica responsável pelo controle, ou seja, por como os dados devem ser manipulados e apresentados ao usuário. As Views decidem qual informação mostrar e como ela deve ser processada antes de ser enviada para a camada de apresentação.
- **Template:** Apresentação visual. Define o layout e a estrutura de como os dados (vindos do Model através da View) serão exibidos para o usuário final. Geralmente, os Templates são arquivos HTML.

DJANGO E O DESENVOLVIMENTO WEB

O Django é considerado um framework full-stack porque ele também oferece ferramentas para lidar com a parte visual da aplicação (templates HTML, CSS, arquivos estáticos), além de toda a lógica de back-end.

É possível construir uma aplicação completa com o Django, sem a necessidade de outros frameworks.



CONFIGURAÇÃO DO AMBIENTE

Antes de inicializar um projeto Django, precisamos criar o ambiente virtual.

Um ambiente virtual é uma ferramenta que permite a criação de ambientes isolados para cada projeto. Cada projeto pode ter suas próprias dependências, evitando conflitos e garantindo que as bibliotecas necessárias estejam disponíveis.

No Python isso é feito por meio da venv que é um um diretório separado que contém uma cópia independente do interpretador Python e um conjunto de pacotes.



COMO INICIAR UM AMBIENTE VIRTUAL

Para criar um ambiente virtual (venv) é só digitar o comando no terminal:

```
python -m venv venv
```

Esse comando deve criar uma pasta chamada venv no diretório.

Para ativar o ambiente virtual temos que usar o seguinte comando:

- Windows: **venv/Scripts/activate**
- Git Bash no Windows: **source venv/Scripts/activate**
- macOS/Linux: **source venv/bin/activate**

INICIALIZANDO UM PROJETO DJANGO

1) Para instalar o Django, com a venv ativada, rode o comando no terminal:

```
pip install django
```

2) Para criar um projeto Django basta rodar o comando no terminal:

```
django-admin startproject nome_do_projeto
```

3) Para rodar o projeto, primeiro acesse a pasta do projeto usando o comando no terminal:

```
cd nome_do_projeto
```

4) E para inicializar o django é só rodar o comando:

```
python manage.py runserver
```


PROJETO X APP

O Django diferencia projeto de App e isso é muito importante na organização do framework.

- **Projeto:** É a estrutura principal que contém as configurações do Django, as URLs gerais e os apps. Um projeto pode conter um ou mais apps, e é onde as configurações globais da aplicação ficam armazenadas.
- **App:** Um app é uma aplicação específica que executa uma função dentro do projeto. É modular e pode ser usada em diferentes projetos. Cada app é responsável por uma parte específica do seu projeto (ex: blog, loja, API).

Então é possível ter um projeto com vários Apps.

ESTRUTURA DO DJANGO

Um projeto Django já inicializa com alguns arquivos. São eles:

- **manage.py**: é um script Python que serve como interface de linha de comando para o seu projeto Django. Ele permite a execução de diversos comandos úteis para o gerenciamento do seu aplicativo, como iniciar o servidor de desenvolvimento, criar e aplicar migrações de banco de dados, coletar arquivos estáticos, entre outras funcionalidades.
- **__init__.py**: indica que a pasta é um módulo Python. Ele não costuma conter código, mas sua presença é necessária para que o Python trate a pasta como um pacote válido. Com isso, você pode importar e organizar o código dentro do projeto.
- **settings.py**: contém todas as configurações essenciais para a execução da sua aplicação web, desde definições de banco de dados até configurações de segurança.
- **urls.py**: gerencia a lógica de roteamento da sua aplicação, dizendo ao Django o que fazer com cada URL que o usuário acessar.
- **wsgi.py e asgi.py**: são responsáveis por permitir que o servidor web e o Django se comuniquem. Isso significa que qualquer requisição que chega ao servidor (como um usuário acessando uma URL) passa por um desses arquivos antes de ser processada pelo Django.

ESTRUTURA DO DJANGO

Além dos arquivos criados automaticamente, o Django segue algumas convenções de organizações de arquivos que são criados a depender da necessidade.

- **views.py**: onde as views são definidas. Elas processam as requisições HTTP e retornam as respostas apropriadas, como páginas HTML, JSON ou outras informações. As views controlam a lógica de negócio da aplicação.
- **models.py**: onde os models são definidos. Eles representam as tabelas no banco de dados e a lógica de dados.
- **admin.py**: configurações para o Django Admin, permitindo que você gerencie os modelos pela interface administrativa.
- **forms.py**: Usado para definir formulários, criando lógica para entrada de dados e validação.
- **templates/**: onde ficam os arquivos HTML usados para renderizar as páginas da aplicação.
- **static/**: usada para armazenar arquivos como CSS, JavaScript e imagens.

HTTP

O protocolo HTTP é a base da comunicação entre o navegador (cliente) e o servidor (Django). Quando o usuário acessa uma URL, o navegador envia uma requisição HTTP para o servidor.

No Django, essa requisição passa por um sistema de URLs, que identifica qual view deve processar a requisição. A view então decide como processar os dados e envia uma resposta HTTP de volta para o navegador.

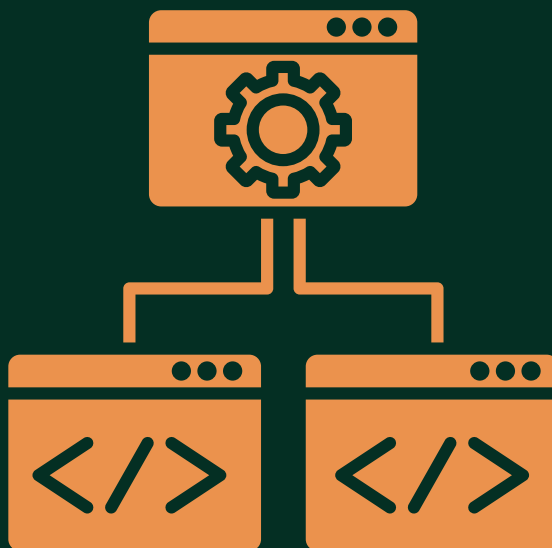
As views podem devolver HTML, JSON ou até mensagens de erro, dependendo da necessidade.



VIEWS

No Django, uma view é uma função Python que recebe uma requisição e retorna uma resposta. As views são o coração da aplicação web, responsáveis por processar a lógica de negócios e gerar o conteúdo a ser exibido para o usuário final.

No Django, as views são organizadas no arquivo **views.py** dentro de cada app, por convenção. Isso mantém o código bem estruturado e facilita a localização das funções ou classes responsáveis por processar as requisições e gerar as respostas.



VIEWS

Para criar uma view, é necessário criar uma função ou uma classe que recebe uma requisição e retorna uma resposta.

É muito comum precisar importar funções do próprio framework para usar na view. Um exemplo comum é importar a função `HttpResponse`, que permite retornar uma resposta HTTP.



```
1  from django.http import HttpResponse
2
3  def index(request):
4      return HttpResponse("Hello, world.")
5
```

URLs

As URLs no Django são responsáveis por definir o caminho de uma requisição. Quando o usuário acessa uma URL, o Django a mapeia para uma view através da URL

As URLs no Django são definidas usando a função `path()`, que mapeia um padrão de URL para uma view. Uma URL pode ter diferentes partes, como diretórios ou parâmetros.

Por convenção, as URLs possuem uma `/` no final no Django. Ela garante que a navegação seja mais consistente e evita erros desnecessários.



```
1  from django.urls import path
2  from .views import index
3
4  urlpatterns = [
5      path('hello/', index),
6  ]
```

ATIVIDADE PRÁTICA

Crie uma lista de dicionários em uma view representando produtos que possuem id, nome, preço e estoque.

Faça uma view para retornar essa lista usando o JsonResponse.

Crie uma nova URL para a view de lista de produtos.

DADOS DINÂMICOS

As aplicações web raramente exibem conteúdo fixo. Em muitos casos, precisamos que a aplicação responda de forma diferente com base em dados fornecidos pelo usuário ou por um banco de dados.

No Django, além de retornar uma resposta simples, podemos passar dados dinâmicos para a view através de parâmetros na URL.

Isso permite que o Django processe diferentes informações com base na entrada do usuário, tornando a aplicação mais interativa e dinâmica.




```
1  from django.http import HttpResponse
2
3
4  def saudation(request, name):
5      return HttpResponse(f"Olá, {name}!")
6
```

DADOS DINÂMICOS

É possível capturar parâmetros dinâmicos diretamente da URL e usá-los dentro das views. Isso permite que a aplicação exiba informações diferentes com base no valor da URL.

Quando mapeamos parâmetros dinâmicos em URLs no Django, é possível especificar o tipo do parâmetro que será capturado na URL. Isso ajuda a garantir que a URL só aceitará dados válidos e do tipo correto.

O tipo do parâmetro é definido entre < e > na URL.



```
1  from django.urls import path
2  from .views import saudation
3
4  urlpatterns = [
5      path('saudacao/<str:name>/', saudation),
6  ]
```

ATIVIDADE PRÁTICA

Crie uma view que recebe como parâmetro um ID de um produto e retorna o produto da lista que contém o mesmo ID.

A view deve buscar o produto na lista de dicionários e retornar todas as informações sobre ele

E deve ser criada uma nova URL que aceite esse parâmetro e chame a view.