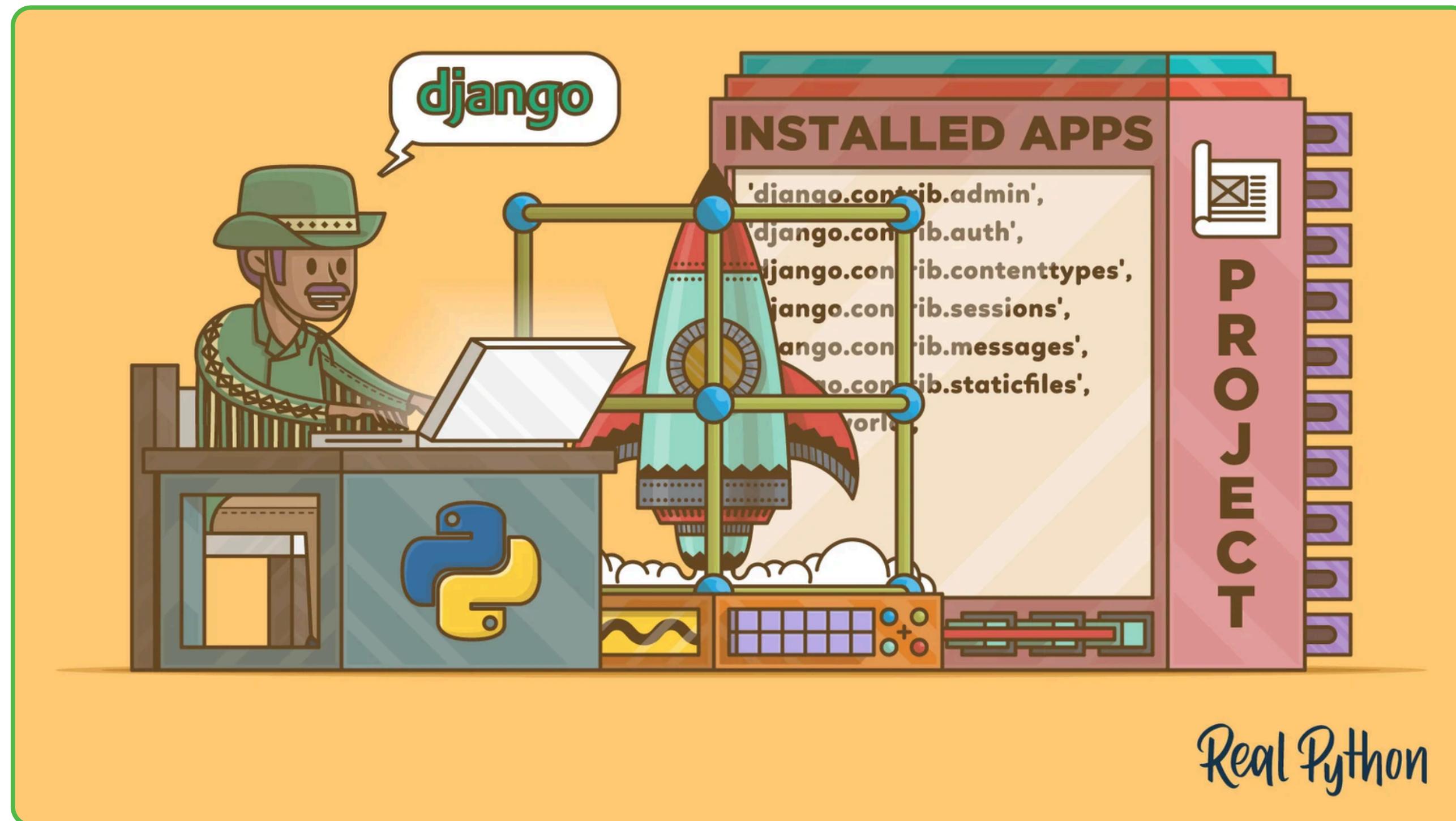


CURSO TÉCNICO EM INFORMÁTICA



DJANGO



CURSO TÉCNICO EM INFORMÁTICA

O que é Django?

Django é um **framework web** de alto nível escrito em Python, projetado para tornar o desenvolvimento de aplicações web mais rápido, limpo e seguro.

Criado por desenvolvedores experientes, ele cuida de boa parte da estrutura necessária para construir um site ou sistema, permitindo que você se concentre na lógica da aplicação em vez de reinventar a roda.

Principais características do Django:

- **Rápido para desenvolver:** com Django, você pode criar um projeto completo em pouco tempo, graças aos seus comandos prontos e estrutura organizada.
- **Seguro por padrão:** o framework já inclui proteções contra ataques comuns como SQL Injection, Cross Site Scripting (XSS) e Cross Site Request Forgery (CSRF).
- **Escalável:** grandes sites como Instagram, Pinterest e Mozilla já utilizaram Django por sua capacidade de lidar com grandes volumes de dados e usuários.
- **Com base no padrão MTV:** Django segue a arquitetura Model-Template-View, semelhante ao MVC, o que ajuda a separar as responsabilidades no código.
- **Painel administrativo automático:** uma das maiores vantagens do Django é gerar, automaticamente, uma interface de administração poderosa e personalizável a partir dos modelos criados.



CURSO TÉCNICO EM INFORMÁTICA

Para que serve?

Django é ideal para criar qualquer tipo de aplicação web, como:

- Blogs
- Sistemas de cadastro
- Plataformas de ensino
- E-commerces
- Painéis administrativos
- APIs REST (com Django REST Framework)



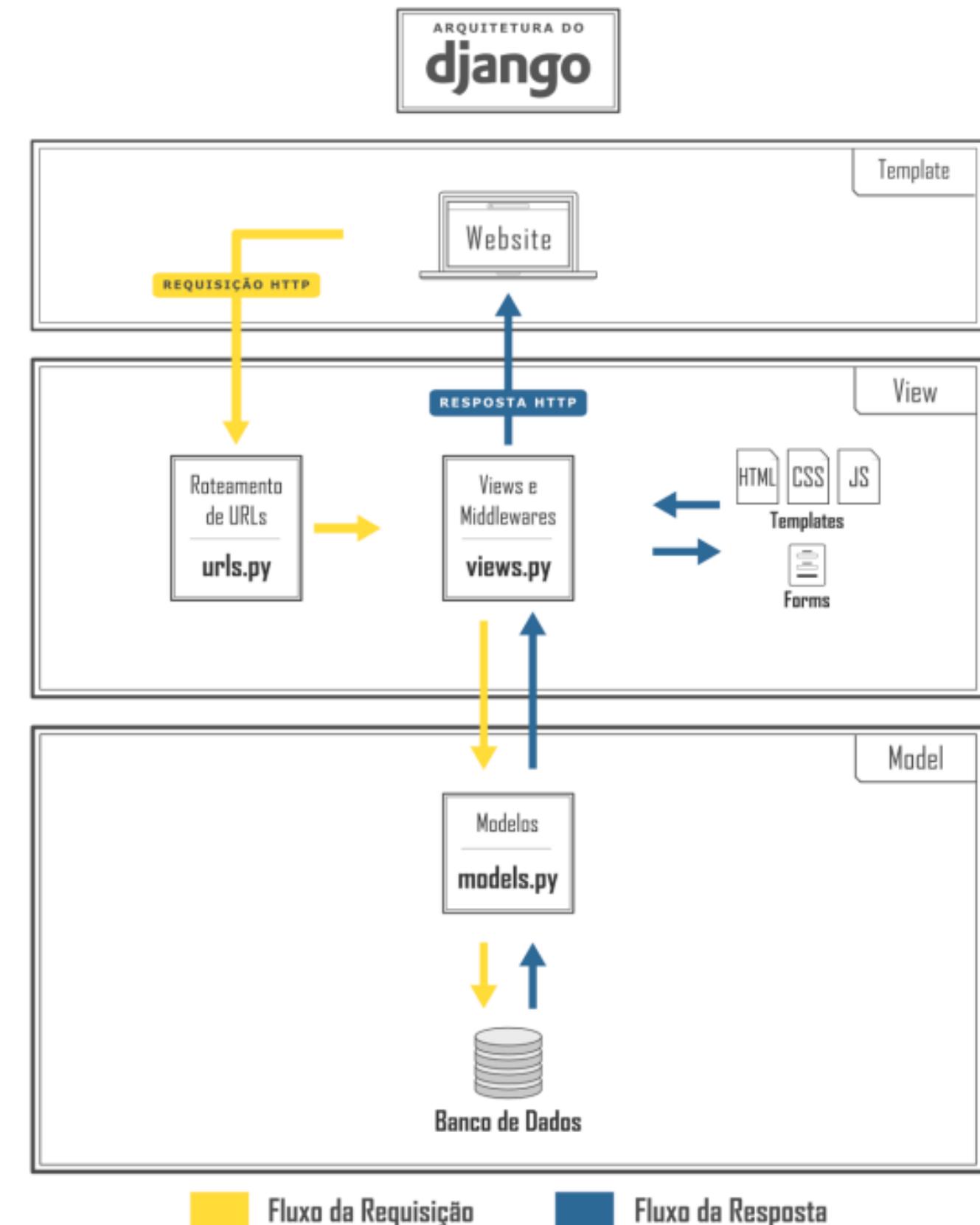
CURSO TÉCNICO EM INFORMÁTICA

Fluxo de Requisição no Django

Quando um usuário acessa uma página em um site feito com Django (por exemplo, digitando uma URL no navegador), ocorre um processo interno chamado fluxo de requisição.

Esse fluxo descreve os passos que o Django segue para receber a requisição HTTP, processá-la e devolver uma resposta ao navegador.

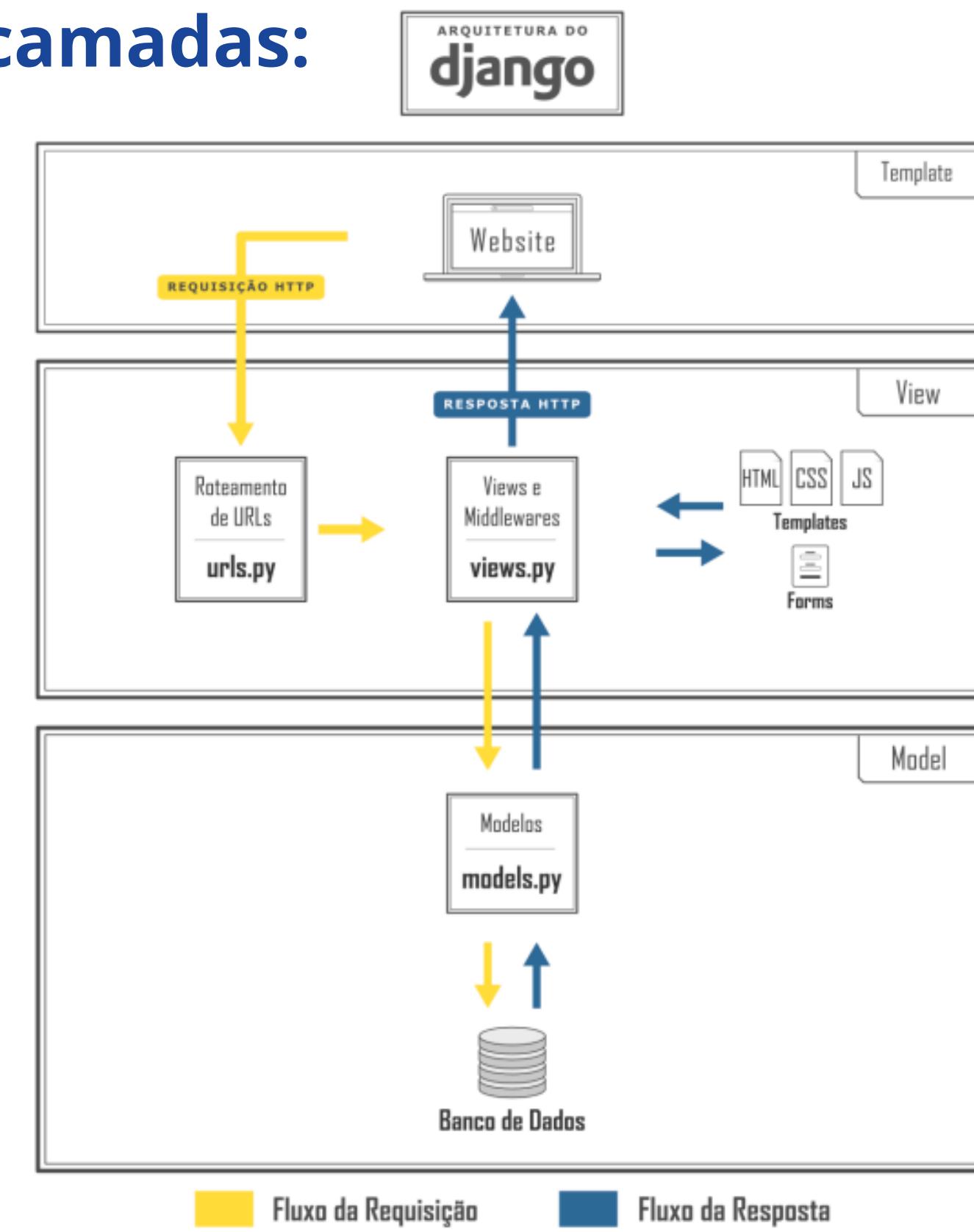
Entender esse fluxo é essencial para saber onde e como escrever seu código de forma organizada e eficiente.



CURSO TÉCNICO EM INFORMÁTICA

O Django é dividido em três camadas:

- A Camada de **Modelos**.
- A Camada de **Views**.
- A Camada de **Templates**.



Etapas do Fluxo de Requisição

O Navegador faz uma requisição HTTP

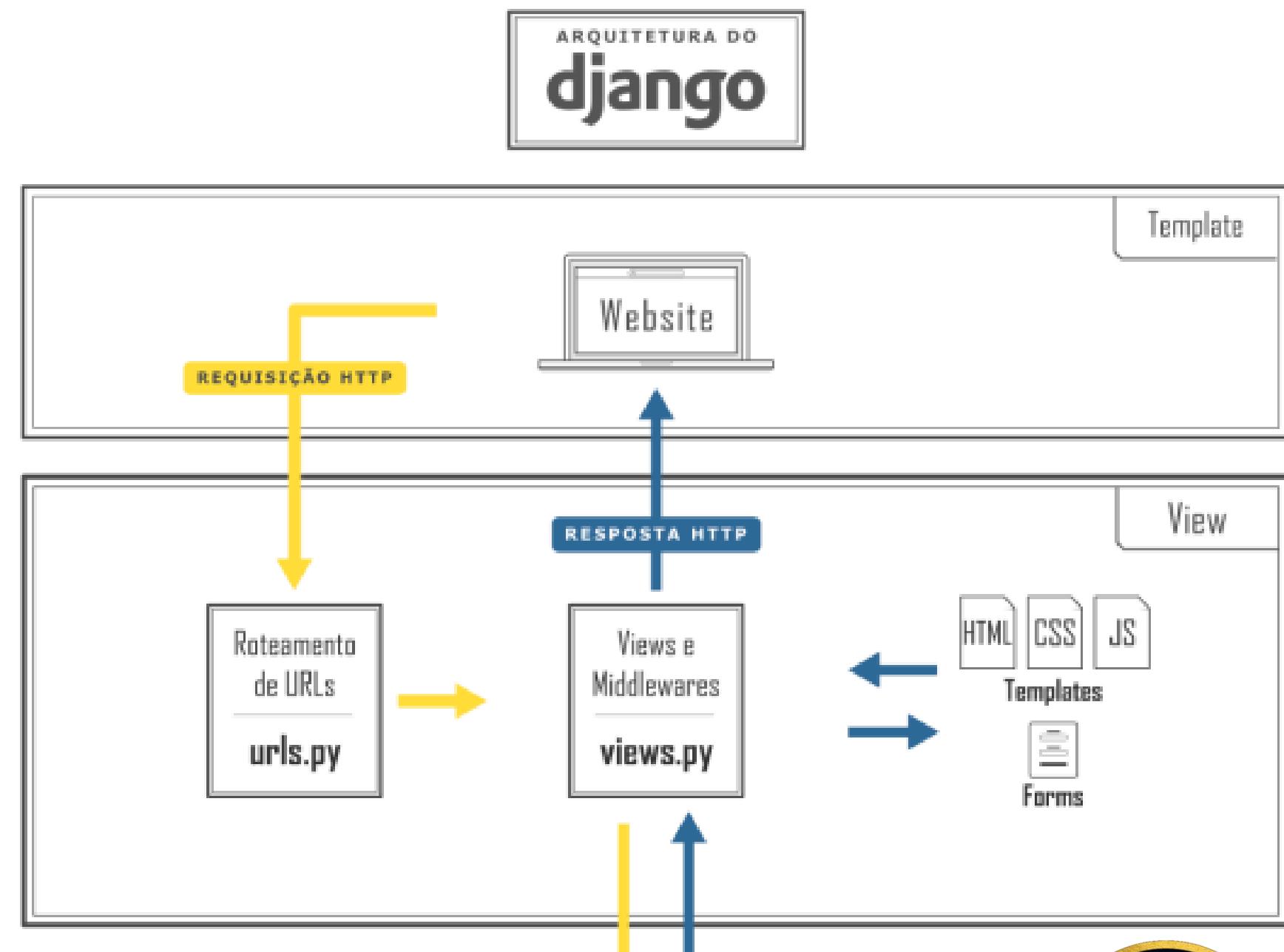
- Tudo começa quando o usuário acessa uma **URL** (ex: <https://meusite.com/alunos/>).
- O navegador envia uma requisição HTTP ao servidor Django.

O Django recebe a requisição via WSGI

- Django é compatível com servidores web através do padrão WSGI (Web Server Gateway Interface). A requisição chega primeiro ao arquivo `wsgi.py`, que repassa essa requisição para o núcleo do Django.

Django busca a URL correspondente (`urls.py`)

- O Django verifica o arquivo **`urls.py`** para encontrar uma rota que corresponda ao caminho requisitado. Se encontrar, ele chama a view associada. Caso contrário, retorna uma resposta de erro 404 (página não encontrada).



Etapas do Fluxo de Requisição

A View é executada (`views.py`)

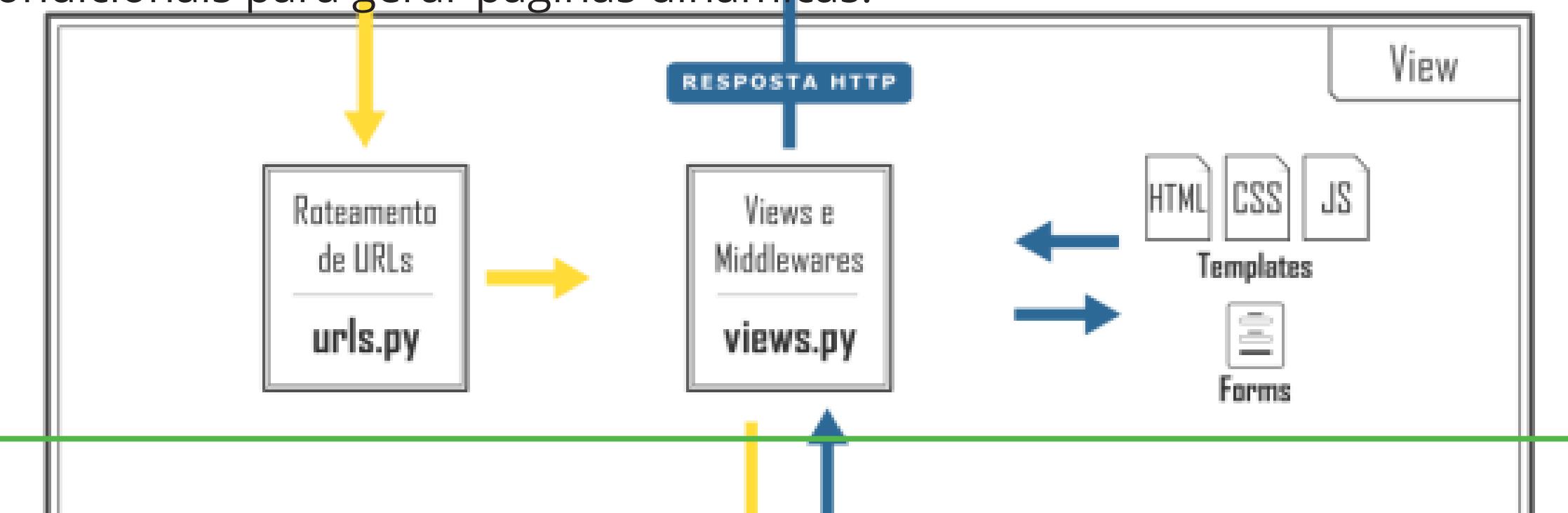
- A **view** é uma função ou classe Python que processa a lógica daquela página. Ela pode consultar o banco de dados, processar formulários, ou simplesmente montar uma resposta HTML.

Modelos podem ser acessados (`models.py`)

- Django é compatível com servidores web através do padrão WSGI (Web Server Gateway Interface). A requisição chega primeiro ao arquivo `wsgi.py`, que repassa essa requisição para o núcleo do Django.

Templates são renderizados (`templates/`)

- A **view** normalmente retorna uma página HTML, que é montada a partir de um arquivo de template. Os templates permitem usar variáveis, laços e condicionais para gerar páginas dinâmicas.

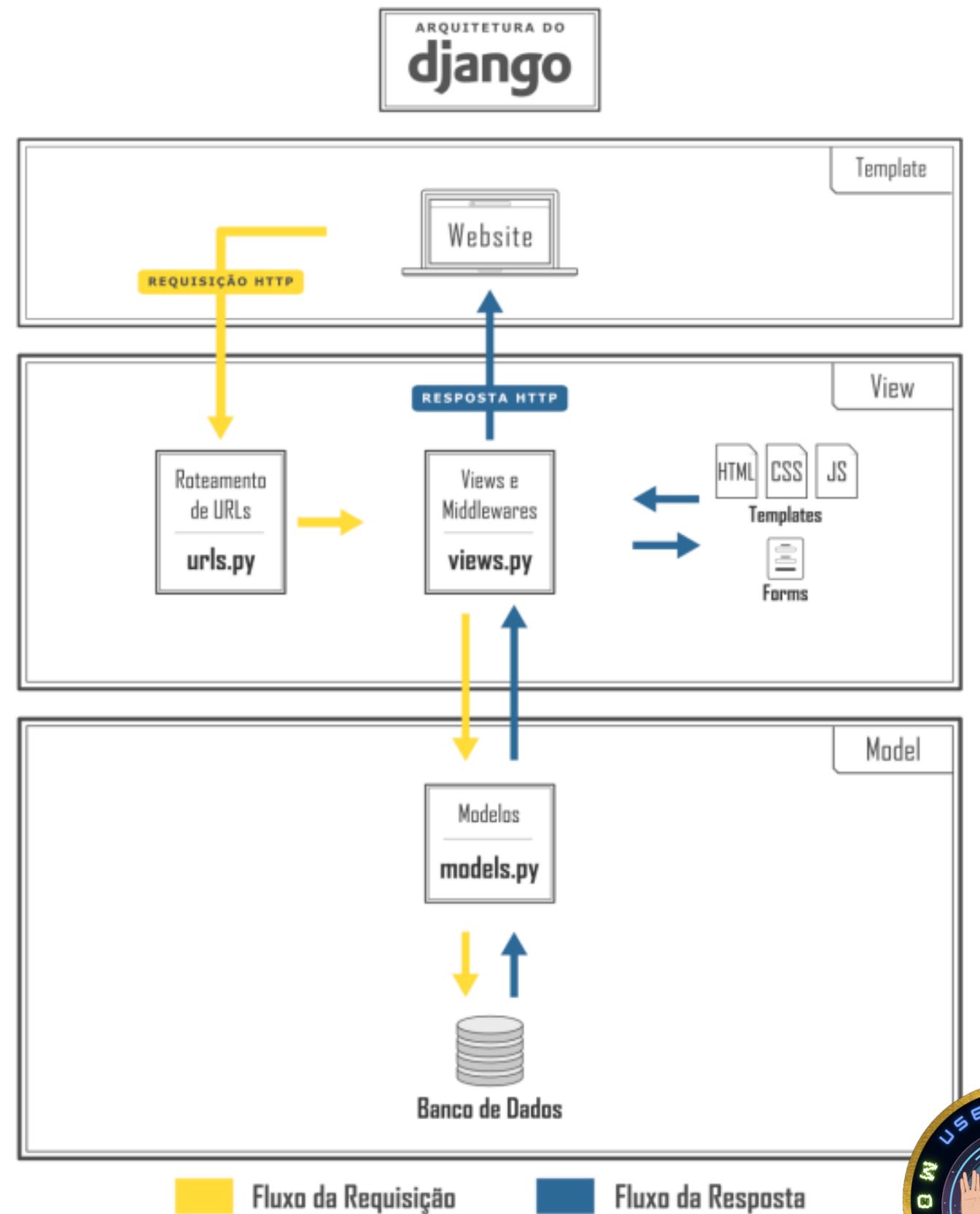


CURSO TÉCNICO EM INFORMÁTICA

Etapas do Fluxo de Requisição

Django retorna uma resposta HTTP

Após a execução da view e renderização do template (se houver), Django envia uma resposta HTTP de volta ao navegador do usuário, finalizando o ciclo.



Fluxo da Requisição

Fluxo da Resposta



CURSO TÉCNICO EM INFORMÁTICA

Preparando o Ambiente e Criando o Projeto Django

Antes de começar a programar com Django, é importante configurar o ambiente de desenvolvimento. Isso garante que as bibliotecas e versões usadas no seu projeto sejam organizadas e independentes de outros projetos **Python** que você possa ter na máquina.

O que é um **virtual environment (venv)**?

É um ambiente isolado onde você pode instalar pacote Python (como o **Django**) sem afetar o restante do sistema.
É uma boa prática sempre criar um venv para cada novo projeto.

Passo 1: Criando o Ambiente Virtual

```
mkdir meuprojeto
```

```
cd meuprojeto
```

Passo 2. Crie o ambiente virtual:

```
python3 -m venv env
```



CURSO TÉCNICO EM INFORMÁTICA

Preparando o Ambiente e Criando o Projeto Django

Passo 2. Crie o ambiente virtual: `python3 -m venv env`

O comando acima cria uma pasta chamada **env** com uma cópia isolada do Python3.

Passo 3. Ative o ambiente virtual: `source env/bin/activate`

Após ativar, o terminal vai mostrar algo como (env) antes do nome do diretório. Isso indica que o ambiente está ativo



CURSO TÉCNICO EM INFORMÁTICA

Instalando o Django e Criando o Projeto

Com o ambiente virtual ativado, instale o Django com o seguinte comando: pip install django

Verifique se instalou corretamente:
django-admin --version

Criando o Projeto Django:

django-admin startproject meuprojeto .

O ponto . no final do comando é importante: ele indica que os arquivos do projeto devem ser criados na pasta atual, e não em uma subpasta separada.

```
meuprojeto/
└── manage.py
└── meuprojeto/
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```



Instalando o Django e Criando o Projeto

- **manage.py**: arquivo de comando do projeto
- **settings.py**: configurações principais (idioma, apps, banco de dados, etc.)
- **urls.py**: onde você define as rotas do seu site
- **wsgi.py e asgi.py**: usados para servir o projeto na web

Rodando o servidor local

python **manage.py runserver**

Depois, abra o navegador e acesse:

<http://127.0.0.1:8000/>

```
meuprojeto/
├── manage.py
└── meuprojeto/
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```



CURSO TÉCNICO EM INFORMÁTICA

Criando um App e Exibindo uma Página com Template

O que é um **app** no Django?

Um app Django é um módulo que possui suas próprias funcionalidades (models, views, templates, URLs etc.) e pode ser reutilizado em outros projetos.

Criando o App

No terminal, dentro da pasta do seu projeto (onde está o manage.py), digite:

python manage.py startapp pagina

Esse comando criará uma nova pasta chamada **pagina/** com a seguinte estrutura:

```
pagina/
└── admin.py
└── apps.py
└── migrations/
└── models.py
└── tests.py
└── views.py
```



CURSO TÉCNICO EM INFORMÁTICA

Registrando o App no Projeto

Abra o arquivo **meuprojeto/settings.py** e, dentro da lista **INSTALLED_APPS**, adicione o nome do novo app:

```
INSTALLED_APPS = [
    # apps internos
    'django.contrib.admin',
    'django.contrib.auth',
    ...
    # app criado
    'pagina',
]
```

Criando a View (a lógica da página)

```
from django.shortcuts import render

def homepage(request):
    return render(request, 'pagina/home.html')
```



CURSO TÉCNICO EM INFORMÁTICA

Criando as URLs do App

Crie um novo arquivo chamado **urls.py** dentro da pasta **página/** e adicione:

```
from django.urls import path
from .views import homepage

urlpatterns = [
    path('', homepage, name='home'),
]
```



Criando as URLs do App

Depois, vá até **meuprojeto/urls.py** e inclua o app nas rotas principais:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('pagina.urls')), # agora a página principal vem do app
]
```



CURSO TÉCNICO EM INFORMÁTICA

Criando o Template HTML

Agora vamos criar a página que será exibida no navegador.

Dentro da pasta do app **pagina/**, crie as seguintes pastas:

```
pagina/  
└ templates/  
    └ pagina/  
        └ home.html
```

Dentro do arquivo **home.html**, escreva o seguinte:

```
<!DOCTYPE html>  
<html lang="pt-br">  
<head>  
    <meta charset="UTF-8">  
    <title>Página Inicial</title>  
</head>  
<body>  
    <h1>Bem-vindo ao meu primeiro site com Django!</h1>  
    <p>Você está visualizando um template HTML renderizado por uma vi-
```



Usando o Contexto da View no Template

Quando criamos uma página em Django, muitas vezes queremos exibir dados dinâmicos (variáveis) no HTML. Para isso, usamos um contexto: um dicionário Python que envia dados da **view** para o template.

O que é contexto?

No Django, o contexto é um dicionário (dict) que contém os dados que serão acessados no HTML através do template.

```
context = {'nome': 'Anderson'}
```

No template:

```
<p>Olá, {{ nome }}!</p>
```



CURSO TÉCNICO EM INFORMÁTICA

Atualize a view em pagina/views.py:

```
from django.shortcuts import render

def homepage(request):
    contexto = {
        'nome': 'Anderson',
        'mensagem': 'Seja bem-vindo ao curso de Django!',
        'linguagens': ['Python', 'JavaScript', 'HTML', 'CSS'],
    }
    return render(request, 'pagina/home.html', contexto)
```



CURSO TÉCNICO EM INFORMÁTICA

Atualize o template pagina/templates/pagina/home.html:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Página com Contexto</title>
</head>
<body>
    <h1>Olá, {{ nome }}!</h1>
    <p>{{ mensagem }}</p>

    <h2>Linguagens estudadas:</h2>
    <ul>
        {% for linguagem in linguagens %}
            <li>{{ linguagem }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

Explicando o que aconteceu:

- A **view** enviou três variáveis para o **template**:
 - nome (uma string)
 - mensagem (outra string)
 - linguagens (uma lista)
- O **template** acessou essas variáveis usando `{{ }}` para valores simples e `{% for %}` para listas.



Usando Condicionais com `{% if %}` nos Templates

Em muitos casos, queremos que o conteúdo exibido em uma página HTML mude conforme os dados recebidos pela view. Por exemplo:

- Mostrar “**Aprovado**” ou “**Reprovado**” com base na média
- Mostrar uma mensagem especial se um campo estiver vazio
- Exibir uma seção apenas para usuários logados

Para isso, o Django fornece tags condicionais no template, como `{% if %}`.

É uma estrutura condicional usada dentro de templates para tomar decisões com base nos dados do contexto.

A sintaxe é simples:

```
{% if condicao %}  
    <!-- HTML exibido se a condição for verdadeira --&gt;<br/>{% endif %}
```



CURSO TÉCNICO EM INFORMÁTICA

Usando Condicionais com `{% if %}` nos Templates

Exemplo Simples: contexto = {'media': 6.5}

código no template:

```
{% if media >= 7 %}  
    <p>Aluno aprovado!</p>  
{% else %}  
    <p>Aluno reprovado.</p>  
{% endif %}
```



CURSO TÉCNICO EM INFORMÁTICA

Também é possível usar elif (else if):

Exemplo Simples: contexto = {'media': 6.5}

código no template:

```
{% if media >= 9 %}  
    <p>Excelente desempenho!</p>  
{% elif media >= 7 %}  
    <p>Aprovado!</p>  
{% else %}  
    <p>Reprovado.</p>  
{% endif %}
```

