Django 101

Autor: Gabriel Felippe

Website: akiradev.netlify.app

Introdução

- Django é um web framework Python de alto nível que permite o desenvolvimento rápido de sites seguros e de fácil manutenção.
- Construído por desenvolvedores experientes, o Django cuida de grande parte do trabalho de desenvolvimento web, para então você poder se concentrar em escrever seu aplicativo sem precisar reinventar a roda.
- É gratuito e de código aberto, tem uma comunidade próspera e ativa, ótima documentação e muitas opções de suporte gratuito e pago.

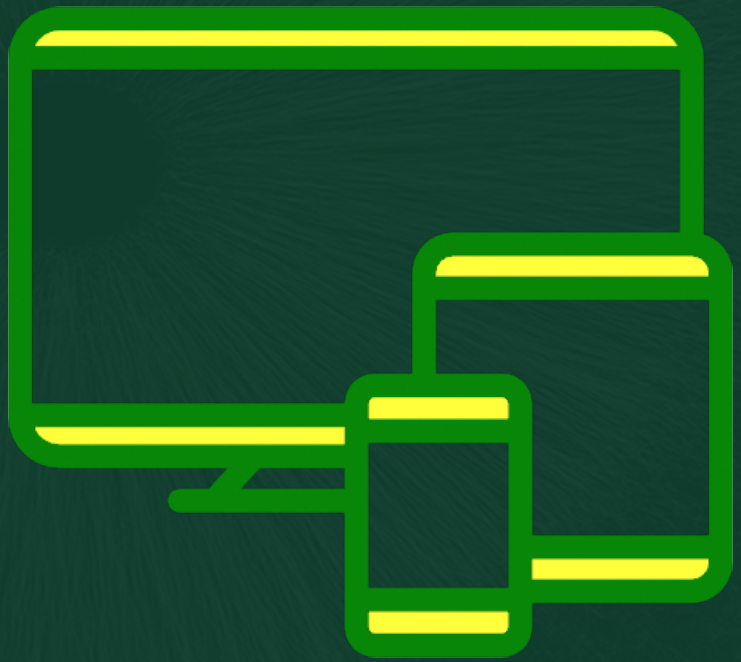
História

- Django foi criado no outono de 2003, quando os programadores web do jornal Lawrence Journal-World, Adrian Holovaty e Simon Willison, começaram a usar Python para construir aplicativos.
- Ele foi lançado publicamente sob uma licença BSD em julho de 2005.

O framework foi nomeado em homenagem ao guitarrista Django Reinhardt.

Django

- Django ajuda você a escrever software que é:
  - Completo
  - Versátil
  - Seguro
  - Escalável
  - Sustentável
  - Portátil



Completo

- Django segue a filosofia "baterias incluídas" e oferece quase tudo que os desenvolvedores desejam fazer "*out of the box*".
- Como tudo que você precisa faz parte de um "produto", tudo funciona perfeitamente em conjunto, segue princípios de design consistentes e tem documentação extensa e atualizada.
- Você pode encontrar a documentação do Django no endereço: https://docs.djangoproject.com/en/stable

Versátil

- Django pode ser (e tem sido) usado para construir quase qualquer tipo de site - desde sistemas de gerenciamento de conteúdo e wikis até redes sociais e sites de notícias.
- Ele pode funcionar com qualquer framework client-side e pode entregar conteúdo em quase qualquer formato (incluindo HTML, RSS feeds, JSON, XML, etc).
- Internamente, embora forneça opções para quase qualquer funcionalidade que você possa desejar (por exemplo, vários bancos de dados populares, templating engines, etc.), ele também pode ser estendido para usar outros componentes, se necessário.

Seguro

- O Django ajuda os desenvolvedores a evitar muitos erros comuns de segurança, fornecendo uma estrutura que foi projetada para "fazer as coisas certas" para proteger o site automaticamente.
- Por exemplo, Django fornece uma maneira segura de gerenciar contas de usuário e senhas, evitando erros comuns como colocar informações de sessão em cookies onde é vulnerável (em vez disso, os cookies contêm apenas uma chave, e os dados reais são armazenados no banco de dados) ou armazenamento direto de senhas em vez de um hash de senha.
- O Django habilita a proteção contra muitas vulnerabilidades por padrão, incluindo SQL injection, cross-site scripting, cross-site request forgery e clickjacking.

Escalável

- O Django usa uma arquitetura baseada em componente “nada compartilhado” (cada parte da arquitetura é independente das outras e pode, portanto, ser substituída ou alterada se necessário).
- Ter uma separação clara entre as diferentes partes significa que ele pode escalar para aumentar o tráfego adicionando hardware em qualquer nível: servidores de cache, servidores de banco de dados ou servidores de aplicação.
- Alguns dos sites mais acessados escalaram o Django com sucesso para atender às suas demandas (por exemplo, Instagram e Disqus).

Sustentável

- O código do Django é escrito usando princípios e padrões de design que encorajam a criação de código sustentável e reutilizável.
- Em particular, ele usa o princípio Don't Repeat Yourself (DRY) para que não haja duplicação desnecessária, reduzindo a quantidade de código.
- O Django também promove o agrupamento de funcionalidades relacionadas em "aplicações" (apps) reutilizáveis e, em um nível inferior, agrupa o código relacionado em módulos, ao longo das linhas do padrão Model View Controller (MVC).

Portátil

- Django é escrito em Python, que roda em muitas plataformas.
- Isso significa que você não está vinculado a nenhuma plataforma de servidor específica e pode executar seus aplicativos em muitos tipos de Linux, Windows e Mac OS X.
- Além disso, o Django é bem suportado por muitos provedores de hospedagem na web, que geralmente fornecem infraestrutura e documentação específicas para hospedar sites Django.

Componentes

- Apesar de ter sua própria nomenclatura, como nomear os objetos que podem ser chamados, gerando as respostas HTTP de "views", o framework Django pode ser visto como uma arquitetura MVC.
- Ele consiste em um object-relational mapper (ORM) que faz a mediação entre modelos de dados (definidos como classes Python) e um banco de dados relacional ("Modelo"), um sistema para processamento de requisições HTTP com um sistema de web templates ("View") e um despachante de URL's ("Controller").

Componentes

- Também estão incluídos na estrutura principal do Django:
  - Um servidor web leve e autônomo para desenvolvimento e teste.
  - Um sistema de serialização e validação de formulários que pode traduzir entre formulários HTML e valores adequados para armazenamento no banco de dados
  - Um sistema de template que utiliza o conceito de herança emprestado da programação orientada a objetos.
  - Uma estrutura de cache que pode usar qualquer um dos vários métodos de cache.
  - Suporte para classes de middleware que podem intervir em vários estágios de processamento de requisição e realizar funções personalizadas.

Componentes

- Também estão incluídos na estrutura principal do Django:
  - Um sistema despachante interno que permite que os componentes de um aplicativo comuniquem eventos uns aos outros por meio de sinais predefinidos.
  - Um sistema de internacionalização, incluindo traduções dos próprios componentes do Django em uma variedade de idiomas.
  - Um sistema de serialização que pode produzir e ler representações XML e/ou JSON de instâncias de modelo Django.
  - Um sistema para estender os recursos do template engine.
  - Uma interface para o Python unit test framework.

Opinativo vs Não-Opinativo

- Os frameworks da Web costumam se referir a si mesmos como "opinativos" ou "não-opinativos".
- Frameworks opinativos são aquelas com opiniões sobre a "maneira certa" de lidar com qualquer tarefa específica. Freqüentemente, eles suportam o desenvolvimento rápido em um domínio específico (resolvendo problemas de um tipo específico) porque a maneira certa de fazer qualquer coisa geralmente é bem compreendida e documentada.
- No entanto, eles podem ser menos flexíveis na resolução de problemas fora de seu domínio principal e tendem a oferecer menos opções para quais componentes e abordagens podem usar.

Opinativo vs Não-Opinativo

- Frameworks não-opinativos, por outro lado, têm muito menos restrições sobre a melhor maneira de unir componentes para atingir uma meta, ou mesmo quais componentes devem ser usados.
- Eles tornam mais fácil para os desenvolvedores usarem as ferramentas mais adequadas para concluir uma tarefa específica, embora ao custo que você precisa encontrar esses componentes por você mesmo.

Opinativo vs Não-Opinativo

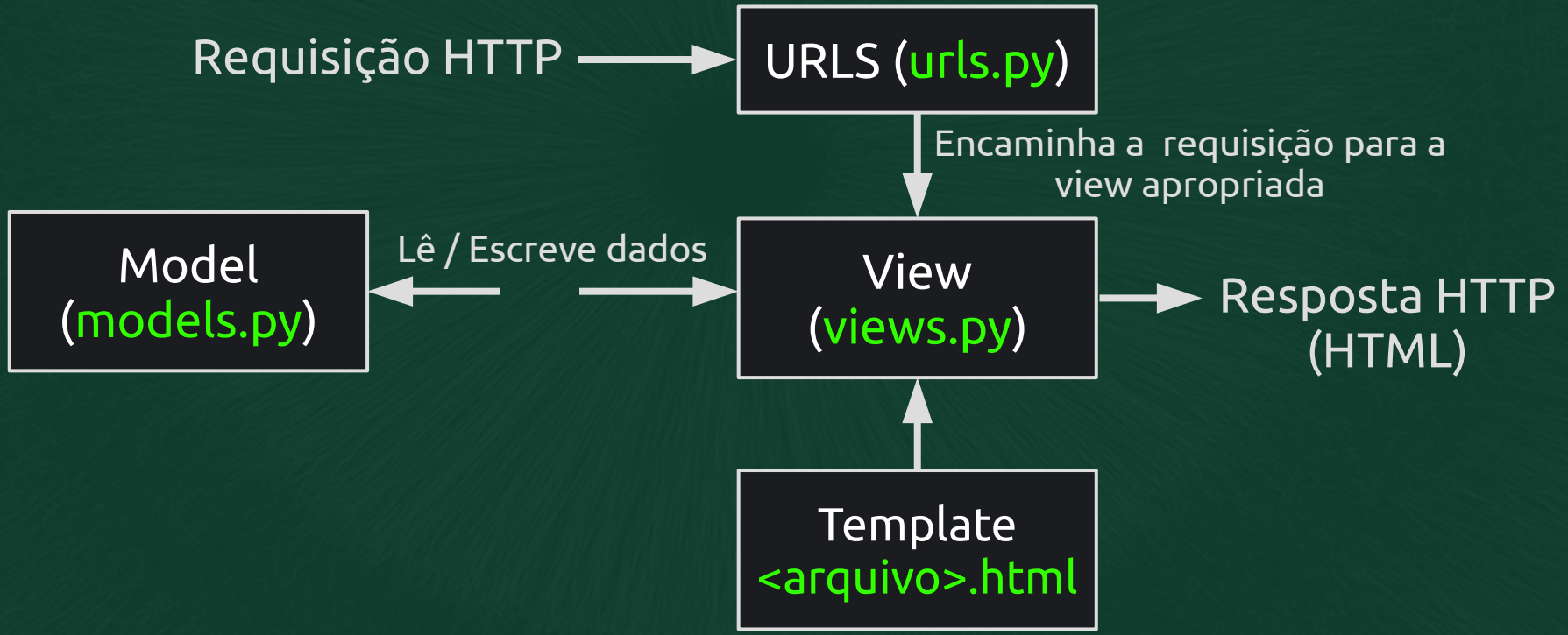
- Django é "um tanto opinativo" e, portanto, oferece o "melhor dos dois mundos".
- Ele fornece um conjunto de componentes para lidar com a maioria das tarefas de desenvolvimento web e uma (ou duas) formas preferidas de usá-los.
- No entanto, a arquitetura desacoplada do Django significa que você geralmente pode escolher entre várias opções diferentes ou adicionar suporte para opções completamente novas, se desejar.

Código Django

- Em um site tradicional baseado em dados, uma aplicação web espera por requisições HTTP do navegador da web (ou outro cliente).
- Quando uma requisição é recebida, o aplicativo calcula o que é necessário com base na URL e, possivelmente, nas informações nos dados POST ou GET.
- Dependendo do que for necessário, ele pode ler ou gravar informações de um banco de dados ou executar outras tarefas necessárias para atender à requisição.
- O aplicativo então retornará uma resposta ao navegador da web, geralmente criando dinamicamente uma página HTML para o navegador exibir, inserindo os dados recuperados em espaços reservados em um template HTML.

Código Django

- As aplicações web Django normalmente agrupam o código que lida com cada uma dessas etapas em arquivos separados:



Código Django

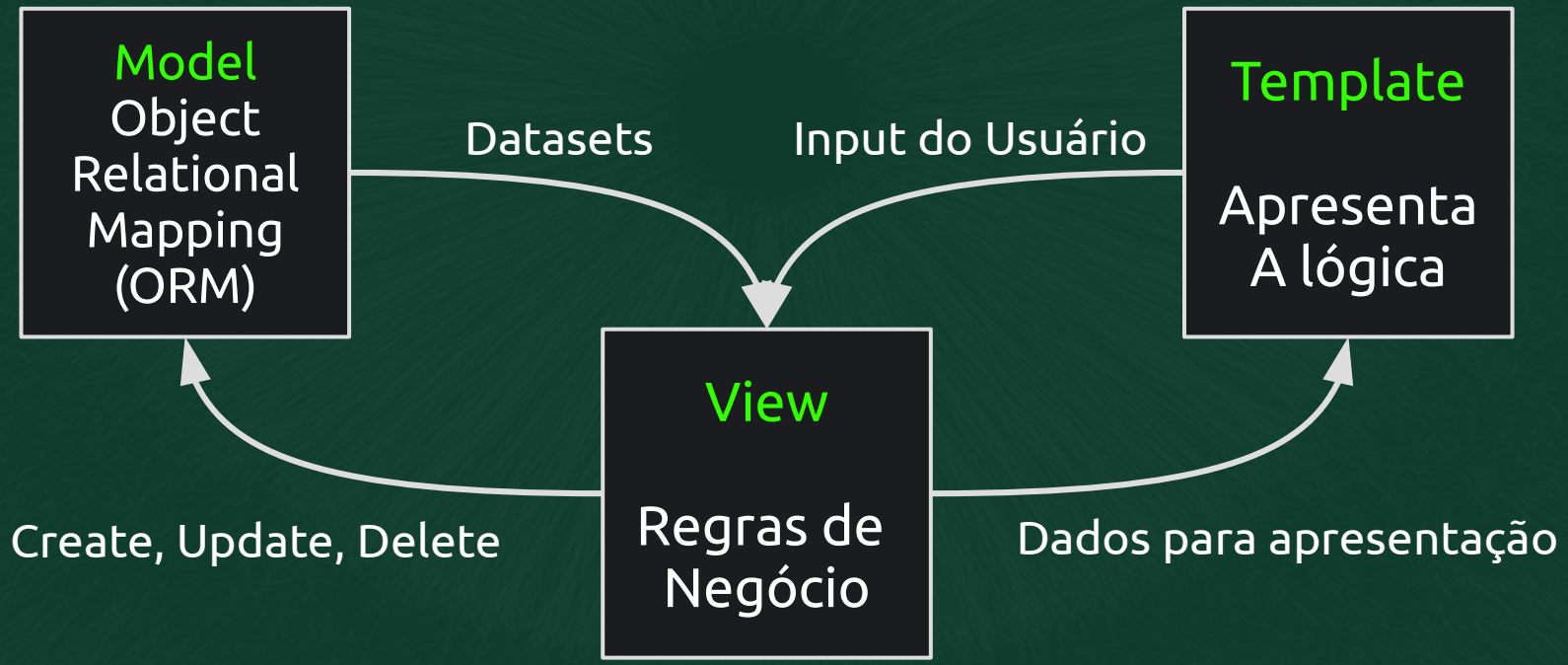
- URLs: embora seja possível processar solicitações de cada URL por meio de uma única função, é muito mais fácil de manter uma aplicação ao escrever uma função de visualização (view) separada para lidar com cada recurso. Um mapeador de URL é usado para redirecionar requisições HTTP para a view apropriada com base na URL de requisição. O mapeador de URL também pode corresponder a padrões específicos de strings ou dígitos que aparecem em uma URL e passá-los para uma função de view como dados.
- View: Uma view é uma função manipuladora de requisição, que recebe requisições HTTP e retorna respostas HTTP. As views acessam os dados necessários para atender às requisições por meio de modelos e delegam a formatação da resposta aos templates.

Código Django

- Models: Os modelos são objetos Python que definem a estrutura dos dados de um aplicativo e fornecem mecanismos para gerenciar (adicionar, modificar, excluir) e consultar registros no banco de dados.
- Templates: Um template é um arquivo de texto que define a estrutura ou layout de um arquivo (como uma página HTML), com placeholders usados para representar o conteúdo real. Uma view pode criar dinamicamente uma página HTML usando um template HTML, preenchendo-a com dados de um modelo. Um template pode ser usado para definir a estrutura de qualquer tipo de arquivo; não precisa ser HTML!

Código Django

- Django se refere a esta organização como a arquitetura "Model View Template (MVT)". Ele tem muitas semelhanças com a arquitetura Model View Controller (MVC) que vimos anteriormente.



Enviando Requisição para uma View

- Um mapeador de URL é normalmente armazenado em um arquivo chamado urls.py. No exemplo abaixo, o mapeador (urlpatterns) define uma lista de mapeamentos entre rotas (padrões de URL específicos) e funções de view correspondentes. Se uma requisição HTTP for recebida com um URL correspondente a um padrão especificado, a função de view associada será chamada e transmitida à requisição.

urls.py

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('book/<int:id>/', views.book_detail, name='book_detail'),
    path('catalog/', include('catalog.urls')),
]
```

O objeto urlpatterns é uma lista de funções path() (as listas Python são definidas usando colchetes, onde os itens são separados por vírgulas e podem ter uma vírgula final opcional. Por exemplo: [item1, item2, item3,]).

Enviando Requisição para uma View

- O primeiro argumento para o método path() é uma rota (padrão) que será correspondida.
- O método path() usa colchetes angulares (<>) para definir partes de uma URL que serão capturadas e passadas para a função de view como argumentos nomeados.
- O segundo argumento é outra função que será chamada quando o padrão for correspondido.
- A notação views.book_detail indica que a função chamada é book_detail() e pode ser encontrada em um módulo chamado views (ou seja, dentro de um arquivo chamado views.py)

Lidando com Requisições (views.py)

- As views são o coração da aplicação web, recebendo requisições HTTP de clientes da web e retornando respostas HTTP.
- No meio disso, eles organizam os outros recursos do framework para acessar bancos de dados, renderizar templates, etc.
- O exemplo abaixo mostra uma função de view mínima index(), que poderia ter sido chamada por nosso mapeador de URL na seção anterior.

views.py

```
from django.http import HttpResponse

def index(request):
    # Get an HttpRequest - the request parameter
    # perform operations using information from the request.
    # Return HttpResponse
    return HttpResponse('Hello from Django!')
```

Lidando com Requisições (views.py)

- Como todas as funções de view, a função index() recebe um objeto HttpRequest como um parâmetro (requisição) e retorna um objeto HttpResponse.
- Nesse caso, não fazemos nada com a requisição e nossa resposta retorna uma string com o texto 'Hello from Django!'.
- É importante lembrar que essa resposta pode ser o conteúdo HTML de uma página Web, ou um redirecionamento, ou um erro 404, ou um documento XML, ou uma imagem. . . ou qualquer coisa, de fato.

Um Pouco sobre Python

- Módulos Python são "bibliotecas" de funções, armazenadas em arquivos separados, que podemos querer usar em nosso código. No exemplo anterior importamos apenas o objeto HttpResponse do módulo django.http para que possamos usá-lo em nossa view: from django.http import HttpResponse. Existem outras maneiras de importar alguns ou todos os objetos de um módulo.
- As funções são declaradas usando a palavra-chave def conforme mostrado acima, com parâmetros nomeados listados entre parênteses após o nome da função; toda a linha termina em dois pontos. Observe como as próximas linhas estão todas recuadas. A indentação é importante, pois especifica que as linhas de código estão dentro desse bloco específico (a indentação obrigatória é um recurso chave do Python e é um dos motivos pelos quais o código Python é tão fácil de ler).

Definindo Modelos de Dados (models.py)

- As aplicações web Django gerenciam e consultam dados por meio de objetos Python chamados de models.
- Os modelos definem a estrutura dos dados armazenados, incluindo os tipos de campo e possivelmente também seu tamanho máximo, valores padrão, opções de lista de seleção, texto de ajuda para documentação, texto de rótulo para formulários, etc.
- A definição do modelo é independente do banco de dados subjacente - você pode escolher um de vários como parte das configurações do projeto.
- Depois de escolher o banco de dados que deseja usar, você não precisa falar com ele diretamente - você apenas escreve a estrutura do seu modelo e código Python, e o Django lida com todo o "trabalho difícil" de comunicação com o banco de dados para vocês.

Definindo Modelos de Dados (models.py)

- O trecho de código a seguir mostra um modelo Django muito simples para um objeto Livro.
- A classe Livro é derivada da classe django models.Model. Ele define o título do livro e o gênero do livro como campos de caracteres e especifica um número máximo de caracteres a serem armazenados para cada registro.
- O genero pode ser um de vários valores, portanto, nós o definimos como um campo de escolha e fornecemos um mapeamento entre as opções a serem exibidas e os dados a serem armazenados, junto com um valor padrão.

Definindo Modelos de Dados (models.py)

- Exemplo de Modelo Django:

models.py

```
from django.db import models

class Livro(models.Model):
    titulo = models.CharField(max_length=40)

    GENEROS = (
        ('Te', 'Terror'),
        ('Fi', 'Filosofia'),
        ('Fa', 'Fantasia'),
        ...  # lista de outros gêneros
    )
    genero = models.CharField(max_length=3, choices=GENEROS, default='U11')
```

Um Pouco sobre Python

- Python suporta "programação orientada a objetos", um estilo de programação em que organizamos nosso código em objetos, que incluem dados relacionados e funções para operar nesses dados.
- Os objetos também podem herdar/estender/derivar de outros objetos, permitindo que um comportamento comum entre objetos relacionados seja compartilhado. Em Python, usamos a palavra-chave class para definir o "projeto" de um objeto. Podemos criar várias instâncias específicas do tipo de objeto com base no modelo da classe.
- Portanto, por exemplo, aqui temos uma classe Livro, que deriva da classe Model. Isso significa que é um modelo e conterá todos os métodos de um modelo, mas também podemos fornecer recursos especializados próprios.
- Em nosso modelo, definimos os campos que nosso banco de dados precisará para armazenar nossos dados, dando-lhes nomes específicos. Django usa essas definições, incluindo os nomes dos campos, para criar o banco de dados subjacente.

Consultando Dados (views.py)

- O modelo Django fornece uma API de consulta simples para pesquisar o banco de dados associado.
- Isso pode corresponder a uma série de campos ao mesmo tempo usando critérios diferentes (por exemplo, exato, não diferenciar maiúsculas de minúsculas, maior que, etc.) e pode oferecer suporte a declarações complexas (por exemplo, você pode especificar uma pesquisa em livros que têm um gênero que começa com "F" ou termina com "ia").
- O trecho de código a seguir mostra uma função de view (manipulador de recursos) para exibir todos os nossos livros de Filosofia.

Consultando Dados (views.py)

- Exemplo de uma view que faz uma consulta de dados:

views.py

```
from django.shortcuts import render
from .models import Livro

def index(request):
    list_livros = Livro.objects.filter(genero__exact="Fi") # Consulta
    context = {'livros_filosofia': list_livros}
    return render(request, '/livros/index.html', context)
```

- A linha comentada mostra como podemos usar a API de consulta de modelo para filtrar todos os registros onde o campo genero tem exatamente o texto 'Fi' (observe como este critério é passado para a função filter() como um argumento, com o nome do campo e o tipo de correspondência separado por um sublinhado duplo: genero__exact).

Consultando Dados (views.py)

- Esta função index() usa a função render() para criar o HttpResponse que é enviado de volta ao navegador.
- Esta função é um atalho; ela cria um arquivo HTML combinando um template HTML especificado e alguns dados para inserir no template (fornecidos na variável chamada "context").
- A seguir, mostramos como o template possui os dados inseridos nele para criar o HTML.

Renderizando Dados (Templates HTML)

- Os sistemas de template permitem que você especifique a estrutura de um documento de output, usando placeholders para dados que serão preenchidos quando uma página for gerada.
- Os templates são freqüentemente usados para criar HTML, mas também podem criar outros tipos de documentos.
- Django suporta seu sistema nativo de templates e outra biblioteca Python popular chamada Jinja2 pronta para uso (ela também pode ser feita para suportar outros sistemas, se necessário).
- O fragmento de código a seguir mostra a aparência do template HTML chamado pela função render() da seção anterior.

Renderizando Dados (Templates HTML)

- Exemplo de Template:

Index.html

```
<html lang="en">
<head>
 <meta charset="utf-8">
 <title>Home page</title>
</head>
<body>
 {% if livros_filosofia %}
  <ul>
   {% for livro in livros_filosofia %}
    <li>{{ livro.titulo }}</li>
   {% endfor %}
  </ul>
 {% else %}
  <p>Nenhum livro disponível.</p>
 {% endif %}
</body>
</html>
```

Renderizando Dados (Templates HTML)

- Este template foi escrito sob a suposição de que terá acesso a uma variável de lista chamada livros_filosofia quando for renderizada (isso está contido na variável de contexto dentro da função render() usada em nossa view).
- Dentro do esqueleto HTML, temos uma expressão que primeiro verifica se a variável livros_filosofia existe e, em seguida, a itera em um for loop. Em cada iteração, o modelo exibe o valor titulo de cada livro selecionado em um elemento <li>.

O Poder do Django

- As seções anteriores mostram as principais características que você usará em quase todos os aplicativos da web: mapeamento de URL, views, modelos e templates.
- Além disso, Django fornece outras opções para trabalharmos:
  - Formulários: formulários HTML são usados para coletar dados do usuário para processamento no servidor. Django simplifica a criação, validação e processamento de formulários.
  - Autenticação e permissões do usuário: Django inclui um sistema robusto de autenticação e permissão do usuário que foi construído com a segurança em mente.

O Poder do Django

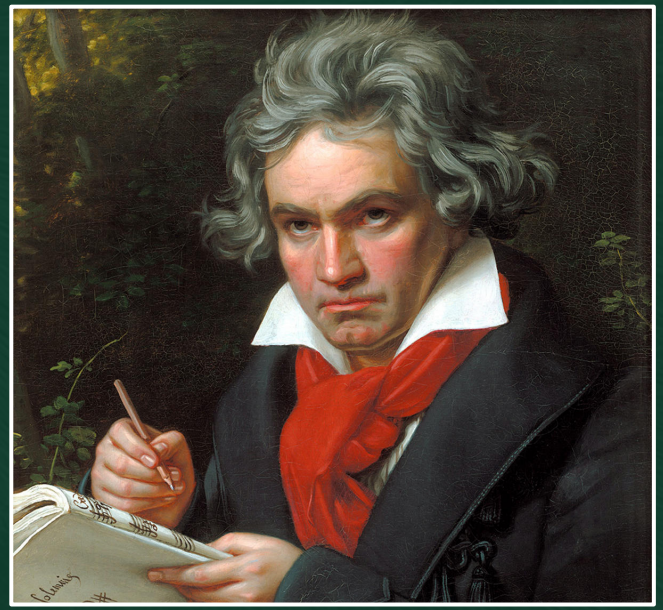
- Cache: a criação de conteúdo dinamicamente é muito mais computacionalmente intensiva (e lenta) do que servir conteúdo estático. Django fornece caching flexível para que você possa armazenar toda ou parte de uma página renderizada para que ela não seja renderizada novamente, exceto quando necessário.
- Site de administração: o site de administração do Django é incluído por padrão quando você cria um aplicativo usando o esqueleto básico. Isso torna muito fácil fornecer uma página de administração para que os administradores do site criem, editem e visualizem quaisquer modelos de dados em seu site.
- Serializando dados: o Django torna fácil serializar e servir seus dados como XML ou JSON. Isso pode ser útil ao criar um serviço da web (um site que fornece dados puramente para serem consumidos por outros aplicativos ou sites, e não exibe nada por si mesmo), ou ao criar um site no qual o código do lado do cliente lida com toda a renderização de dados.

Tutoriais

- Agora que você está familiarizado com os conceitos do framework Django, os dois tutoriais a seguir vão te ensinar como desenvolver um Blog:
- Básico:
  - https://github.com/the-akira/CC33Z/blob/master/Cursos/Django%20101/Tutorial.md
- Intermediário:
  - https://akiradev.netlify.app/posts/django-blog-heroku

Considerações Finais

“Don’t only practice your art, but force your way into its secrets, for it and knowledge can raise men to the divine.” Ludwig van Beethoven



Referências

- https://www.djangoproject.com/
- https://en.wikipedia.org/wiki/Django_(web_framework)
- https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction