FAP UFRN 2024 – Prof. José Alfredo Costa

Outubro de 2024

Material para estudos e desenvolvimento com Exemplos baseados no curso de Django – Disponível em <https://www.youtube.com/watch?v=p5MCJLIn_is>

Ebook: Introdução ao Django - Aula 1: Configurando o Ambiente

Bem-vindos à primeira aula do nosso curso de Django! Nesta aula, vamos preparar o terreno para construir aplicações web incríveis. **Mãos à massa!**

**Pré-requisitos:**

* Python instalado
* Editor de código de sua preferência (ex: VS Code)

**Criando o Projeto:**

1. **Pasta do Projeto:** Crie uma pasta para o projeto. Vamos chamá-la de "learn\_log".
2. **Ambiente Virtual:**
   * Abra o terminal e navegue até a pasta "learn\_log".
   * Crie um ambiente virtual com o comando:

python -m venv ll\_env

**ll\_env** é o nome do ambiente. Use outro nome se preferir.

* + Ative o ambiente virtual:
  + .\ll\_env\Scripts\activate (Windows)

source ll\_env/bin/activate (Linux/macOS)

* + - **Observação:** O comando source pode não funcionar em algumas versões do Windows. Se encontrar problemas, use o comando .\ll\_env\Scripts\activate.
    - Veja que o nome do ambiente virtual aparece entre parênteses no terminal após a ativação.

1. **Instalando o Django:**
   * Com o ambiente virtual ativo, execute:

pip install Django

* + Verifique a versão do Django:

django-admin --version

1. **Criando o Projeto Django:**
   * Execute o comando:

django-admin startproject learning\_log .

* + - O ponto final (.) indica que o projeto será criado na pasta atual ("learn\_log").

1. **Abrindo o Projeto no Editor de Código:**
   * Navegue até a pasta do projeto no terminal e execute:

code .

 Isso abrirá a pasta "learn\_log" no VS Code. Se você usa outro editor, abra a pasta manualmente.

**Entendendo a Estrutura:**

* **manage.py:** Ferramenta de linha de comando para gerenciar o projeto.
* **Pasta "learning\_log":** Contém o projeto em si.
  + **init.py:** Arquivo vazio que indica que a pasta é um pacote Python.
  + **settings.py:** Configurações do projeto (banco de dados, etc.).
  + **urls.py:** Define os padrões de URL da aplicação.
  + **wsgi.py:** Interface para servidores web servirem o projeto.
* **ll\_env:** Ambiente virtual do projeto.

**Criando o Banco de Dados:**

* No terminal, execute:

python manage.py migrate

* + Isso criará o banco de dados SQLite padrão do Django.

**Executando o Servidor:**

* No terminal, execute:

python manage.py runserver

* Acesse [http://127.0.0.1:8000/](https://www.google.com/url?sa=E&q=http%3A%2F%2F127.0.0.1%3A8000%2F) no navegador para ver o servidor em execução.

**Parabéns!** Você configurou com sucesso um ambiente de desenvolvimento Django. Na próxima aula, começaremos a construir nossa aplicação.

**Considerações Finais:**

Lembre-se de ativar o ambiente virtual sempre que for trabalhar no projeto. Utilize este guia como referência durante o curso. Até a próxima!

Ebook: Introdução ao Django - Aula 2: Criando Modelos

Nesta aula, vamos aprender a criar **modelos**, a forma como o Django representa as tabelas do nosso banco de dados usando Python. A vantagem? Nada de SQL!

**Visualizando o Banco de Dados:**

* Faça o download e instale o **DB Browser for SQLite**. Ele permite visualizar e gerenciar bancos de dados SQLite.
* Abra o arquivo db.sqlite3 localizado na pasta raiz do seu projeto no DB Browser. Por enquanto, veremos apenas as tabelas padrão do Django.

**Criando o App:**

1. **Ative o ambiente virtual.**
2. No terminal, execute o comando para criar um novo app:

python manage.py startapp learnings

learnings (no plural) é uma convenção para o nome do app principal.

1. **Inclua o novo app em settings.py:**
   * Abra o arquivo settings.py dentro da pasta learning\_log.
   * Localize a variável INSTALLED\_APPS.
   * Adicione 'learnings' à lista, como mostrado abaixo:
   * INSTALLED\_APPS = [
   * # ... outras apps ...
   * 'learnings',

]

**Definindo Modelos:**

* Abra o arquivo models.py dentro da pasta learnings.
* Crie um modelo chamado Topic que representará um tópico de aprendizado:

from django.db import models

class Topic(models.Model):

"""Um assunto sobre o qual o usuário está aprendendo."""

text = models.CharField(max\_length=200)

date\_added = models.DateTimeField(auto\_now\_add=True)

def \_\_str\_\_(self):

"""Devolve uma representação em string do modelo."""

return self.text

**Entendendo o Código:**

* **from django.db import models:** Importa a classe base Model.
* **class Topic(models.Model):** Define o modelo Topic, herdando de models.Model.
* **text = models.CharField(max\_length=200):** Um campo de texto para o nome do tópico, com tamanho máximo de 200 caracteres.
* **date\_added = models.DateTimeField(auto\_now\_add=True):** Um campo de data e hora que registra automaticamente quando o tópico foi adicionado.
* **def \_\_str\_\_(self):** Define como o objeto Topic será representado como string (ex: no painel administrativo).

**Criando Migrations:**

* As migrations são arquivos que dizem ao Django como atualizar o banco de dados com base nas mudanças nos modelos. Para criá-las, execute no terminal:

python manage.py makemigrations learnings

* + O argumento learnings indica ao Django para procurar por mudanças nos modelos dentro do app learnings.

**Aplicando Migrations:**

* Execute o seguinte comando para aplicar as migrations e criar a tabela no banco de dados:

python manage.py migrate

 .

**Verificando o Banco de Dados:**

* Abra novamente o banco de dados no DB Browser e veja que a tabela learnings\_topic foi criada com os campos id, text e date\_added.

**Próximos Passos:**

Na próxima aula, vamos acessar o **painel administrativo** do Django e usar o modelo Topic para adicionar dados ao banco de dados. Continue praticando!

Ebook: Introdução ao Django - Aula 3: Painel Administrativo e Relacionamentos

Nesta aula, vamos explorar o poderoso **painel administrativo** do Django, criar um superusuário para acessá-lo e aprender a conectar nossos modelos através de **relacionamentos**.

**Acessando o Painel Administrativo:**

1. **Execute o Servidor:** Certifique-se de que o servidor Django está em execução com python manage.py runserver.
2. **Acesse a URL:** Abra seu navegador e acesse http://127.0.0.1:8000/admin/. Você verá a tela de login do painel.

**Criando o Superusuário:**

* Ainda no terminal, execute o comando:

python manage.py createsuperuser

* Siga as instruções para definir um nome de usuário, endereço de e-mail e senha. **Lembre-se de guardar essas informações!**

**Registrando o Modelo Topic:**

* Para que o Topic apareça no painel, precisamos registrá-lo.
* Abra o arquivo admin.py dentro da pasta learnings.
* Importe o modelo Topic e registre-o:

from django.contrib import admin

from .models import Topic # Importe o modelo

admin.site.register(Topic) # Registre o modelo

**Atualize o navegador:** A página do painel administrativo agora exibirá "Topics".

**Criando o Modelo Entry:**

* Vamos criar um modelo para representar as anotações (entradas) relacionadas a um tópico.
* Abra o arquivo models.py dentro da pasta learnings e adicione o seguinte código:

from django.db import models

# ... (código do modelo Topic) ...

class Entry(models.Model):

"""Algo específico aprendido sobre um assunto."""

topic = models.ForeignKey(Topic, on\_delete=models.CASCADE)

text = models.TextField()

date\_added = models.DateTimeField(auto\_now\_add=True)

class Meta:

verbose\_name\_plural = 'entries'

def \_\_str\_\_(self):

"""Devolve uma representação em string do modelo."""

return f"{self.text[:50]}..."

**Detalhes do Código:**

* **topic = models.ForeignKey(Topic, on\_delete=models.CASCADE):** Cria uma relação de chave estrangeira entre Entry e Topic.
  + Cada Entry pertence a um Topic.
  + on\_delete=models.CASCADE indica que, ao excluir um Topic, todas as Entries relacionadas também serão excluídas.
* **text = models.TextField():** Campo de texto para a anotação, sem limite de caracteres.
* **class Meta:** Usada para configurar opções do modelo.
  + verbose\_name\_plural = 'entries': Define o plural correto para exibição no painel.
* **\_\_str\_\_:** Retorna os primeiros 50 caracteres da anotação.

**Criando e Aplicando Migrations:**

* No terminal, execute os comandos:
* python manage.py makemigrations learnings

python manage.py migrate

**Registrando o Modelo Entry:**

* Abra o arquivo admin.py dentro da pasta learnings novamente.
* Importe e registre o modelo Entry:

from django.contrib import admin

from .models import Topic, Entry

admin.site.register(Topic)

admin.site.register(Entry)

* **Atualize o navegador:** Agora você verá "Entries" no painel administrativo.

**Testando o Relacionamento:**

1. **Adicione alguns tópicos** através do painel administrativo.
2. **Acesse "Entries"** e adicione novas entradas, **selecionando o tópico** ao qual cada entrada pertence.

**Excluindo Tópicos:**

* Ao tentar excluir um tópico que possui entradas relacionadas, o Django exibirá um aviso, confirmando a exclusão em cascata.

**Conclusão:**

Nesta aula, você aprendeu a:

* Acessar e usar o painel administrativo do Django.
* Criar um superusuário.
* Registrar modelos para gerenciá-los no painel.
* Criar um relacionamento de chave estrangeira entre modelos.

Explore o painel, experimente adicionar e editar dados! Na próxima aula, vamos continuar aprimorando nosso projeto.

Ebook: Introdução ao Django - Aula 4: Interagindo com o Shell Interativo

Nesta rápida aula, vamos explorar o **shell interativo do Django**, uma ferramenta poderosa para testar nossos modelos e interagir com o banco de dados diretamente do terminal.

**Para que serve o Shell Interativo?**

* **Testar modelos:** Criar, recuperar, atualizar e excluir objetos dos seus modelos.
* **Experimentar consultas:** Testar diferentes maneiras de buscar dados no banco de dados.
* **Depurar código:** Inspecionar objetos e variáveis durante o desenvolvimento.

**Acessando o Shell:**

1. **Ative o ambiente virtual.**
2. No terminal, dentro da pasta do projeto, execute:

python manage.py shell

 .

**Importando Modelos:**

* Importe os modelos que você deseja usar no shell:

from learnings.models import Topic, Entry

**Exemplos de Uso:**

* **Recuperando todos os tópicos:**

topics = Topic.objects.all()

for topic in topics:

print(topic.id, topic.text)

* **Recuperando um tópico específico pelo ID:**

topic = Topic.objects.get(id=1)

print(topic.text)

* **Acessando atributos do modelo:**

print(topic.date\_added)

* **Acessando entradas relacionadas a um tópico:**

entries = topic.entry\_set.all()

for entry in entries:

print(entry.text)

**Vantagens do Shell Interativo:**

* **Agilidade:** Permite testar código rapidamente sem precisar executar o servidor.
* **Feedback imediato:** Você vê os resultados das suas ações no banco de dados em tempo real.
* **Aprendizado:** Uma ótima forma de explorar os seus modelos e as APIs do Django.

**Quando usar o Shell Interativo?**

* **Fase de desenvolvimento:** Para testar modelos e experimentar com código.
* **Depuração:** Para inspecionar objetos e entender o comportamento do seu código.
* **Tarefas administrativas:** Para realizar operações pontuais no banco de dados.

**Lembre-se:**

* Utilize o shell interativo principalmente durante o desenvolvimento.
* Para tarefas mais complexas ou que serão executadas repetidamente, crie scripts Python ou utilize o painel administrativo.

**Saindo do Shell:**

* Digite exit() ou pressione Ctrl+D.

Experimente os exemplos e explore as funcionalidades do shell! Na próxima aula, começaremos a construir as páginas da nossa aplicação.

Ebook: Introdução ao Django - Aula 5: Introdução ao HTML e CSS

Antes de mergulharmos na construção de páginas web dinâmicas com Django, precisamos de uma base sólida em **HTML** e **CSS**, as linguagens que dão estrutura e estilo às páginas web.

**HTML: A Estrutura da Web**

* HTML significa *HyperText Markup Language* (Linguagem de Marcação de Hipertexto).
* É uma linguagem de **marcação**, não de programação, o que significa que define a estrutura e o conteúdo da página.
* Utiliza **tags** para marcar elementos como títulos, parágrafos, imagens e links.

**Estrutura Básica de um Documento HTML:**

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="UTF-8">

<title>Título da Página</title>

</head>

<body>

<!-- Conteúdo da página -->

</body>

</html>

 .Html

* **<!DOCTYPE html>:** Define o tipo de documento como HTML5.
* **<html lang="pt-br">:** Define o idioma do documento (importante para acessibilidade e SEO).
* **<head>:** Contém informações sobre a página, como título, links para CSS e metadados.
  + **<title>:** Define o título que aparece na aba do navegador.
  + **<meta charset="UTF-8">:** Define o conjunto de caracteres para suportar caracteres especiais (como acentos).
* **<body>:** Contém todo o conteúdo visível da página, como texto, imagens e links.

**Tags Importantes:**

* **Títulos:** <h1>, <h2>, <h3>, <h4>, <h5>, <h6> (h1 é o maior, h6 o menor)
* **Parágrafo:** <p>
* **Divisão:** <div> (usada para agrupar elementos)
* **Link:** <a href="https://www.exemplo.com">Texto do link</a>

**CSS: Estilizando a Web**

* CSS significa *Cascading Style Sheets* (Folhas de Estilo em Cascata).
* É usado para definir a aparência da página, como cores, fontes, espaçamento e layout.

**Adicionando CSS a um Documento HTML:**

* **Dentro de uma tag <style>:**

<head>

<style>

body {

background-color: lightblue;

}

</style>

</head>

 .Html

* **Importando um arquivo CSS externo (recomendado):**

<head>

<link rel="stylesheet" href="style.css">

</head>

 .Html

**Sintaxe CSS:**

seletor {

propriedade: valor;

propriedade: valor;

}

 .Css

* **Seletor:** Indica qual elemento HTML será estilizado (ex: body, h1, .minha-classe).
* **Propriedade:** A característica que você deseja modificar (ex: background-color, font-size).
* **Valor:** O valor atribuído à propriedade (ex: blue, 16px).

**Classes e IDs:**

* **Classes:** Usadas para aplicar estilos a vários elementos HTML.
* **IDs:** Usados para aplicar estilos a um único elemento HTML.

**Exemplo de Classe:**

<div class="minha-classe">Conteúdo da div</div>

 .Html

.minha-classe {

color: red;

}

 .Css

**Exemplo de ID:**

<div id="minha-div">Conteúdo da div</div>

 .Html

#minha-div {

background-color: yellow;

}

 .Css

**Dicas:**

* Utilize um editor de código com destaque de sintaxe para HTML e CSS.
* Explore a documentação do HTML e CSS para conhecer mais tags e propriedades.
* Utilize ferramentas de desenvolvimento do navegador (tecla F12) para inspecionar e modificar o código HTML e CSS em tempo real.

**Próximos Passos:**

Com essa base em HTML e CSS, você está pronto para começar a criar páginas web com Django! Nas próximas aulas, vamos unir o poder do Django com a flexibilidade do HTML e CSS.

Ebook: Introdução ao Django - Aula 6: Criando Rotas e Nossa Primeira Página HTML

Chegou a hora de dar vida ao nosso projeto Django, construindo a primeira página que os usuários verão! Nesta aula, vamos aprender a:

* **Definir rotas:** Como o Django direciona as requisições dos usuários para as funções corretas.
* **Criar views:** Funções Python que processam as requisições e geram o conteúdo da página.
* **Renderizar templates:** Usar arquivos HTML para exibir o conteúdo dinamicamente.

**Entendendo o Arquivo urls.py:**

* O arquivo urls.py na raiz do projeto (learning\_log/urls.py) é o mapa de rotas do Django.
* Ele mapeia URLs para views, ou seja, define qual função Python será executada quando um usuário acessar uma determinada URL.

**Configurando a Rota Principal:**

1. **Importe include:**

from django.urls import path, include

* + include é usado para incluir rotas de outros arquivos urls.py (geralmente dentro dos apps).

1. **Mapeie a URL vazia:**

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('learnings.urls')), # Nova linha

]

 path('', include('learnings.urls')) diz ao Django: "Se a URL for vazia (apenas o domínio), procure as rotas no arquivo urls.py do app learnings".

**Criando o Arquivo urls.py do App:**

1. **Crie o arquivo:** Crie um novo arquivo chamado urls.py dentro da pasta do seu app (learnings).
2. **Importe path e a view index:**

from django.urls import path

from . import views

1. **Defina a rota para index:**

urlpatterns = [

path('', views.index, name='index'),

]

path('', views.index, name='index') mapeia a URL vazia para a função index dentro do arquivo views.py.

* + name='index' atribui um nome à rota, o que será útil para referenciá-la no código.

**Criando a View index:**

1. **Abra o arquivo views.py** dentro da pasta do app learnings.
2. **Crie a função index:**

from django.shortcuts import render

def index(request):

"""A página inicial de Learning Log."""

return render(request, 'learnings/index.html')

* + render(request, 'learnings/index.html') renderiza o template index.html e envia o resultado para o usuário.

**Criando o Template index.html:**

1. **Crie a pasta templates:** Dentro do seu app learnings, crie uma pasta chamada templates.
2. **Crie a pasta do app:** Dentro da pasta templates, crie outra pasta com o mesmo nome do app (learnings).
3. **Crie o arquivo index.html:** Dentro da pasta learnings/templates/learnings, crie um arquivo chamado index.html.
4. **Adicione conteúdo HTML:**

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="UTF-8">

<title>Learning Log</title>

</head>

<body>

<h1>Learning Log</h1>

<p>Essa aplicação te ajudará a organizar seus estudos.</p>

</body>

</html>

 .Html

**Executando o Projeto:**

1. **Reinicie o servidor Django.**
2. **Acesse a URL raiz:** Abra seu navegador e acesse http://127.0.0.1:8000/.

**Parabéns!** Você criou sua primeira página web dinâmica com Django!

**Explicando o Fluxo:**

1. **Requisição:** O usuário digita a URL raiz no navegador.
2. **Roteamento:** O Django consulta o urls.py do projeto e encontra a rota para a URL vazia.
3. **include:** O Django é direcionado para o arquivo urls.py do app learnings.
4. **index:** A rota vazia no urls.py do app learnings leva à função index.
5. **Processamento:** A função index processa a requisição.
6. **Renderização:** render usa o template index.html para gerar a página HTML final.
7. **Resposta:** O Django envia a página HTML para o navegador do usuário.

**Próximos Passos:**

Com essa estrutura básica, você pode começar a expandir seu projeto, adicionando mais views, templates e funcionalidades. Continue explorando o poder do Django!

Ebook: Introdução ao Django - Aula 7: Herança de Templates e Links Internos

Nesta aula, vamos aprender a otimizar a construção de páginas web com Django, utilizando a **herança de templates** e **criando links internos**.

**Problema:**

Imagine um site com várias páginas (Home, Sobre, Contato). É comum que elementos como o menu, cabeçalho e rodapé se repitam em todas as páginas. Sem herança de templates, teríamos que copiar e colar esses elementos em cada arquivo HTML, o que tornaria o código repetitivo e difícil de manter.

**Solução: Herança de Templates**

* **Template Base:** Um template que contém a estrutura básica do site, com os elementos que se repetem em todas as páginas.
* **Templates Filhos:** Templates que herdam do template base e definem apenas o conteúdo específico de cada página.

**Exemplo Prático:**

1. **Crie o Template Base (base.html):**

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="UTF-8">

<title>Learning Log</title>

</head>

<body>

<header>

<h1><a href="{% url 'index' %}">Learning Log</a></h1>

</header>

<main>

{% block content %}

{% endblock content %}

</main>

</body>

</html>

 .Html

* + **{% url 'index' %}:** Gera a URL da rota com o nome 'index' (definida no urls.py).
  + **{% block content %}{% endblock content %}:** Define um bloco chamado "content" onde os templates filhos inserirão seu conteúdo.

1. **Crie o Template Filho (index.html):**

{% extends 'learnings/base.html' %}

{% block content %}

<p>Essa aplicação te ajudará a organizar seus estudos.</p>

{% endblock content %}

 .Html

* + **{% extends 'learnings/base.html' %}:** Indica que o template index.html herda do template base.
  + **{% block content %}:** Insere o conteúdo do parágrafo dentro do bloco "content" definido no template base.

**Vantagens da Herança de Templates:**

* **Reutilização de código:** Evite repetir o código da estrutura básica em todas as páginas.
* **Manutenção simplificada:** Alterações no template base refletem em todas as páginas que o herdam.
* **Organização:** Separa a estrutura do site do conteúdo específico de cada página.

**Criando Links Internos:**

* Utilize a tag {% url 'nome\_da\_rota' %} para gerar URLs para outras views do seu projeto.
* No exemplo acima, {% url 'index' %} gera a URL para a view index, que está mapeada para a URL raiz.

**Configurando as Rotas:**

* **No urls.py do projeto:** Inclua as rotas do app learnings com include('learnings.urls').
* **No urls.py do app learnings:** Defina a rota para a view index com path('', views.index, name='index').
  + O name='index' define o nome da rota, usado na tag {% url %}.

**Fluxo de Execução:**

1. **Requisição:** O usuário acessa a URL raiz.
2. **Roteamento:** O Django encontra a rota para a URL raiz e chama a view index.
3. **Renderização:** A view index renderiza o template index.html.
4. **Herança:** O template index.html herda do base.html, incluindo o cabeçalho com o link para a página inicial.
5. **Resposta:** O Django envia a página HTML completa para o navegador.

**Conclusão:**

A herança de templates e a criação de links internos são técnicas essenciais para construir sites Django de forma eficiente e organizada. Utilize-as para criar sites mais complexos com facilidade!

Ebook: Introdução ao Django - Aula 8: Blocos, Loops e Contextos

Nesta aula, vamos aprofundar nosso conhecimento sobre templates Django, explorando:

* **Entendendo block e endblock:** Como organizar e reutilizar trechos de código HTML.
* **Utilizando for em templates:** Como exibir dados dinamicamente a partir de um loop.
* **Passando contexto para templates:** Como enviar dados do banco de dados para as views e, em seguida, para os templates.

**O Poder dos Blocos (block e endblock)**

* **Organização:** Blocos delimitam áreas no template base que podem ser personalizadas pelos templates filhos.
* **Reutilização:** Permitem que diferentes templates filhos insiram conteúdo em locais específicos do template base, sem precisar repetir toda a estrutura.

**Exemplo:**

* **base.html:**

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="UTF-8">

<title>Learning Log</title>

</head>

<body>

<header>

<h1><a href="{% url 'index' %}">Learning Log</a></h1>

</header>

<main>

{% block content %}

{% endblock content %}

</main>

<footer>

{% block footer %}

<p>Copyright © 2023 Learning Log</p>

{% endblock footer %}

</footer>

</body>

</html>

 .Html

* **index.html:**

{% extends 'learnings/base.html' %}

{% block content %}

<p>Bem-vindo ao Learning Log!</p>

{% endblock content %}

 .Html

* Neste exemplo, o template index.html define apenas o conteúdo do bloco "content". O cabeçalho, rodapé e a estrutura HTML geral são herdados do base.html.
* Podemos ter quantos blocos forem necessários e nomeá-los de acordo com o conteúdo que receberão.

**Laços de Repetição (for) em Templates**

* A tag {% for %} permite iterar sobre listas e exibir dados dinamicamente.
* Para cada item da lista, o código dentro do bloco {% for %} será executado.

**Exemplo:**

<h2>Tópicos:</h2>

<ul>

{% for topic in topics %}

<li>{{ topic.text }}</li>

{% empty %}

<li>Não há tópicos adicionados.</li>

{% endfor %}

</ul>

 .Html

* Este código itera sobre uma lista chamada topics, que deve ser passada para o template pela view.
* Para cada tópico na lista, um item de lista (<li>) é criado, exibindo o texto do tópico.
* A cláusula {% empty %} exibe uma mensagem caso a lista topics esteja vazia.

**Passando Contexto para Templates**

* As views são responsáveis por buscar dados do banco de dados e passá-los para os templates.
* Isso é feito através de um **dicionário de contexto**.

**Exemplo:**

* **views.py:**

from django.shortcuts import render

from .models import Topic

def topics(request):

"""Mostra todos os assuntos."""

topics = Topic.objects.order\_by('date\_added')

context = {'topics': topics}

return render(request, 'learnings/topics.html', context)

* **topics.html:**

{% extends 'learnings/base.html' %}

{% block content %}

<h2>Tópicos:</h2>

<ul>

{% for topic in topics %}

<li>{{ topic.text }}</li>

{% empty %}

<li>Não há tópicos adicionados.</li>

{% endfor %}

</ul>

{% endblock content %}

 .Html

* Neste exemplo, a view topics busca os tópicos do banco de dados, ordena-os por data e cria um dicionário de contexto com a chave topics e o valor da lista de tópicos.
* O template topics.html usa a variável topics dentro do loop {% for %} para exibir os tópicos.

**Fluxo de Execução:**

1. **Requisição:** O usuário acessa a URL correspondente à view topics.
2. **Processamento:** A view topics busca os dados, cria o contexto e renderiza o template topics.html.
3. **Renderização:** O template topics.html herda do base.html, insere a lista de tópicos no bloco "content" e exibe a página.

**Conclusão:**

Combinando blocos, loops e contextos, podemos criar templates dinâmicos que exibem dados do banco de dados de forma organizada e eficiente. Continue explorando as ferramentas de template do Django para construir aplicações web cada vez mais complexas!

Ebook: Introdução ao Django - Aula 9: Criando Páginas de Tópicos com Anotações

Nesta aula, vamos dar um grande passo na construção do nosso projeto, aprendendo a:

* **Criar páginas dinâmicas:** Mostrar um tópico específico com suas anotações, com base no ID do tópico na URL.
* **Buscar dados pelo ID:** Usar o método get() para recuperar um objeto do banco de dados com base em seu ID.
* **Relacionar modelos em views:** Acessar as anotações relacionadas a um tópico através da chave estrangeira.
* **Formatar datas em templates:** Usar filtros para exibir datas em um formato amigável.

**Configurando a Rota Dinâmica:**

1. **Abra o arquivo urls.py** dentro do app learnings.
2. **Adicione a rota para topic:**

from django.urls import path

from . import views

urlpatterns = [

# ... outras rotas ...

path('topics/<int:topic\_id>/', views.topic, name='topic'),

]

* + path('topics/<int:topic\_id>/', views.topic, name='topic') define uma rota dinâmica que captura um número inteiro (<int:topic\_id>) da URL e o passa como argumento para a view topic.
  + topic\_id é o nome da variável que receberá o ID do tópico.
  + name='topic' define o nome da rota para uso posterior.

**Criando a View topic:**

1. **Abra o arquivo views.py** dentro do app learnings.
2. **Crie a função topic:**

from django.shortcuts import render

from .models import Topic, Entry

def topic(request, topic\_id):

"""Mostra um único assunto e todas as suas entradas."""

topic = Topic.objects.get(id=topic\_id)

entries = topic.entry\_set.order\_by('-date\_added')

context = {'topic': topic, 'entries': entries}

return render(request, 'learnings/topic.html', context)

* + topic = Topic.objects.get(id=topic\_id): Busca o tópico com o ID especificado na URL.
  + entries = topic.entry\_set.order\_by('-date\_added'): Busca as entradas relacionadas ao tópico, ordenadas pela data em ordem decrescente (mais recentes primeiro).
  + O dicionário de contexto context contém o tópico e as entradas relacionadas.

**Criando o Template topic.html:**

1. **Crie o arquivo topic.html** dentro da pasta learnings/templates/learnings.
2. **Adicione o código HTML:**

{% extends 'learnings/base.html' %}

{% block content %}

<h3>{{ topic.text }}</h3>

<p>Entradas:</p>

<ul>

{% for entry in entries %}

<li>

<p>{{ entry.date\_added|date:"d M Y, H:i" }}</p>

<p>{{ entry.text|linebreaks }}</p>

</li>

{% empty %}

<li>Não há registros.</li>

{% endfor %}

</ul>

{% endblock content %}

 .Html

* + {{ topic.text }}: Exibe o texto do tópico.
  + {% for entry in entries %}: Itera sobre as entradas relacionadas ao tópico.
  + {{ entry.date\_added|date:"d M Y, H:i" }}: Formata a data da entrada usando o filtro date.
  + {{ entry.text|linebreaks }}: Insere quebras de linha no texto da entrada usando o filtro linebreaks.

**Testando a Página:**

1. **Reinicie o servidor Django.**
2. **Acesse a URL:** http://127.0.0.1:8000/topics/3/ (substitua 3 pelo ID de um tópico existente).

**Fluxo de Execução:**

1. **Requisição:** O usuário acessa a URL com o ID do tópico.
2. **Roteamento:** O Django encontra a rota dinâmica e chama a view topic, passando o ID como argumento.
3. **Processamento:** A view busca o tópico e suas entradas, cria o contexto e renderiza o template topic.html.
4. **Renderização:** O template exibe o tópico e suas entradas formatadas.
5. **Resposta:** O Django envia a página HTML para o navegador.

**Conclusão:**

Parabéns! Você aprendeu a criar páginas dinâmicas que exibem dados específicos do banco de dados. Continue explorando os recursos do Django para construir aplicações web cada vez mais poderosas!

Ebook: Introdução ao Django - Aula 10: Links de Navegação Entre Páginas

Nesta aula, vamos aprimorar a navegação do nosso site, transformando os tópicos listados na página topics.html em links que levam às suas respectivas páginas de detalhes.

**Ajustes Iniciais:**

* **base.html:** Remova a tag <h1> do template base, pois ela será incluída no template topics.html.
* **urls.py (app learnings):** Altere a rota dinâmica para path('topics/<int:topic\_id>/', views.topic, name='topic'), para manter a consistência com o restante do código.
* **base.html:** Adicione um link para a página topics.html no cabeçalho:

<header>

<h1><a href="{% url 'index' %}">Learning Log</a></h1>

<a href="{% url 'topics' %}">Tópicos</a>

</header>

 .Html

**Criando Links para as Páginas de Tópicos:**

1. **Abra o arquivo topics.html.**
2. **Dentro do loop {% for %}, envolva o texto do tópico com a tag <a>:**

{% extends 'learnings/base.html' %}

{% block content %}

<h2>Tópicos:</h2>

<ul>

{% for topic in topics %}

<li>

<a href="{% url 'topic' topic.id %}">{{ topic.text }}</a>

</li>

{% empty %}

<li>Não há tópicos adicionados.</li>

{% endfor %}

</ul>

{% endblock content %}

 .Html

* + **{% url 'topic' topic.id %}:** Gera a URL para a view topic, passando o ID do tópico atual como argumento.
  + Agora, cada tópico na lista será um link que leva à sua página de detalhes.

**Entendendo o Fluxo:**

1. **Página topics.html:** O loop {% for %} itera sobre a lista de tópicos, e para cada tópico:
   * A tag {% url 'topic' topic.id %} gera a URL correta para a página do tópico, usando o ID do tópico.
   * A tag <a> cria o link com o texto do tópico.
2. **Clique no Link:** Quando o usuário clica no link de um tópico:
   * O navegador é redirecionado para a URL gerada (ex: /topics/2/).
   * O Django captura o ID do tópico da URL (ex: 2).
   * O Django chama a view topic, passando o ID como argumento.
3. **View topic:**
   * A view busca o tópico com o ID especificado.
   * A view busca as entradas relacionadas ao tópico.
   * A view renderiza o template topic.html, passando o tópico e as entradas no contexto.
4. **Página topic.html:**
   * O template exibe os detalhes do tópico e suas entradas.

**Conclusão:**

Agora, nosso site possui links de navegação dinâmicos, permitindo que os usuários acessem as páginas de detalhes de cada tópico com facilidade. Essa estrutura torna a navegação mais intuitiva e organizada. Continue explorando as funcionalidades do Django para criar sites cada vez mais interativos!

Ebook: Introdução ao Django - Aula 11: Formulários e Autenticação de Usuários (Parte 1)

Nesta aula, vamos dar um passo crucial na construção de aplicações web robustas com Django: **formulários**. Eles são a ponte entre o usuário e o servidor, permitindo que os usuários interajam com o sistema e enviem dados para serem processados e armazenados. Nosso objetivo final? Permitir que cada usuário tenha sua própria conta e gerencie seus tópicos individualmente.

**Por que usar Formulários no Django?**

* **Simplicidade:** O Django facilita a criação, validação e processamento de formulários, com menos código e maior segurança.
* **Validação Integrada:** O Django verifica automaticamente os dados enviados pelos usuários, garantindo que sejam válidos e seguros.
* **Integração com Modelos:** Formulários podem ser facilmente criados a partir de modelos, simplificando a inserção e atualização de dados no banco de dados.

**Criando um Formulário para Novos Tópicos:**

**Crie o arquivo forms.py:** Dentro do app learnings, crie um arquivo chamado forms.py.

**Importe as classes necessárias:**

from django import forms

from .models import Topic

**Defina a classe TopicForm:**

class TopicForm(forms.ModelForm):

class Meta:

model = Topic

fields = ['text']

labels = {'text': ''}

* + **forms.ModelForm:** Indica que este formulário será baseado em um modelo.
  + **model = Topic:** Define o modelo que será usado como base para o formulário.
  + **fields = ['text']:** Especifica os campos do modelo que serão incluídos no formulário.
  + **labels = {'text': ''}:** Remove o rótulo padrão do campo text.

**Criando a View new\_topic:**

**Abra o arquivo views.py** dentro do app learnings.

**Crie a função new\_topic:**

from django.shortcuts import render, redirect

from django.http import HttpResponseRedirect

from django.urls import reverse

from .models import Topic

from .forms import TopicForm

def new\_topic(request):

"""Adiciona um novo assunto."""

if request.method != 'POST':

# Nenhum dado submetido; cria um formulário em branco.

form = TopicForm()

else:

# Dados de POST submetidos; processa os dados.

form = TopicForm(request.POST)

if form.is\_valid():

form.save()

return redirect('learnings:topics')

context = {'form': form}

return render(request, 'learnings/new\_topic.html', context)

* + **if request.method != 'POST'**: Verifica se a requisição é do tipo POST. Se não for, significa que o formulário está sendo exibido pela primeira vez, então um formulário em branco é criado.
  + **else**: Se a requisição for POST, os dados do formulário são processados.
    - **form = TopicForm(request.POST)**: Cria um formulário com os dados enviados na requisição.
    - **if form.is\_valid()**: Valida os dados do formulário. Se forem válidos:
      * **form.save()**: Salva os dados no banco de dados.
      * **return redirect('learnings:topics')**: Redireciona o usuário para a página de tópicos.
  + **context = {'form': form}**: Cria um dicionário de contexto com o formulário.
  + **return render(request, 'learnings/new\_topic.html', context)**: Renderiza o template new\_topic.html com o formulário.

**Criando a Rota new\_topic:**

**Abra o arquivo urls.py** dentro do app learnings.

**Adicione a rota para new\_topic:**

from django.urls import path

from . import views

app\_name = 'learnings'

urlpatterns = [

# ... outras rotas ...

path('new\_topic/', views.new\_topic, name='new\_topic'),

]

* + **path('new\_topic/', views.new\_topic, name='new\_topic')**: Mapeia a URL /new\_topic/ para a view new\_topic.

**Na próxima aula, criaremos o template new\_topic.html e finalizaremos a implementação do formulário.**

Ebook: Introdução ao Django - Aula 12: Formulários e Autenticação de Usuários (Parte 2)

Dando continuidade à aula anterior, vamos criar o template para o formulário de novos tópicos e entender como o Django lida com o envio de dados via POST e GET.

**Criando o Template new\_topic.html:**

1. **Crie o arquivo new\_topic.html** dentro da pasta learnings/templates/learnings.
2. **Adicione o código HTML:**

{% extends 'learnings/base.html' %}

{% block content %}

<form action="{% url 'learnings:new\_topic' %}" method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button name="submit" type="submit" class="btn btn-primary">Adicionar Novo Tópico</button>

</form>

{% endblock content %}

**Explicando o Código:**

* **{% extends 'learnings/base.html' %}:** Herda do template base.
* **<form action="{% url 'learnings:new\_topic' %}" method="post">:** Define o formulário HTML.
  + **action="{% url 'learnings:new\_topic' %}":** Define a URL para onde os dados do formulário serão enviados (a mesma URL da view new\_topic).
  + **method="post":** Especifica que os dados serão enviados via método POST.
* **{% csrf\_token %}:** Token de segurança do Django para proteger contra ataques CSRF (Cross-Site Request Forgery).
* **{{ form.as\_p }}:** Renderiza o formulário como parágrafos (<p>), formatando cada campo como um bloco separado.
* **<button name="submit">:** Cria o botão de envio do formulário.

**Entendendo POST e GET:**

* **GET:** Usado para **buscar** dados do servidor. Os dados são enviados na URL, visíveis na barra de endereço. Exemplos:
  + Acessar uma página: http://127.0.0.1:8000/topics/
  + Buscar um tópico específico: http://127.0.0.1:8000/topics/3/
* **POST:** Usado para **enviar** dados para o servidor, geralmente para **modificar** dados. Os dados são enviados no corpo da requisição, invisíveis na URL. Exemplos:
  + Enviar um formulário de contato.
  + Criar um novo tópico.

**Fluxo de Execução:**

1. **Página new\_topic.html:** O formulário é exibido.
2. **Envio do Formulário:** Quando o usuário clica no botão "Adicionar Novo Tópico":
   * Os dados do formulário são enviados para o servidor via método POST.
   * A URL de destino é a mesma da view new\_topic.
3. **View new\_topic:**
   * O Django identifica que a requisição é do tipo POST.
   * O Django cria um formulário TopicForm com os dados enviados.
   * O Django valida os dados do formulário.
   * Se os dados forem válidos, o Django salva o novo tópico no banco de dados.
   * O Django redireciona o usuário para a página topics.

**Vantagens de Reutilizar a View:**

* **Menos código:** Não precisamos criar uma view separada para processar o formulário.
* **Simplicidade:** A lógica para exibir e processar o formulário fica concentrada em um único lugar.

**Testando o Formulário:**

1. **Reinicie o servidor Django.**
2. **Acesse a URL:** http://127.0.0.1:8000/new\_topic/
3. **Preencha o formulário** e clique em "Adicionar Novo Tópico".
4. **Verifique** se o novo tópico foi adicionado à lista na página topics.

**Conclusão:**

Com a implementação do formulário, os usuários agora podem adicionar novos tópicos ao sistema. Na próxima aula, vamos aprender a criar formulários para adicionar novas entradas (anotações) aos tópicos.