FAP UFRN 2024 – Prof. José Alfredo Costa

Outubro de 2024 V2

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

José Alfredo F. Costa – Outubro 2024

**Capítulo 1: Introdução ao Django e Configuração do Ambiente**

**1.1 O que é Django?**

Antes de mergulharmos na prática, é fundamental entender o que é Django e por que ele é uma escolha tão popular para o desenvolvimento web. Django é um framework web de alto nível escrito em Python que permite a construção rápida de aplicativos web robustos e escaláveis. Ele oferece muitas funcionalidades "out of the box", como:

* Um sistema de autenticação robusto.
* Um painel administrativo completo.
* Suporte a banco de dados via ORM (Object-Relational Mapping).
* Ferramentas para criar URLs amigáveis.
* Segurança integrada contra falhas comuns, como injeção SQL e Cross-Site Scripting (XSS).

Django segue o padrão **MVT (Model-View-Template)**, que se assemelha ao famoso MVC (Model-View-Controller). No MVT, o **Model** representa os dados da aplicação, o **View** lida com a lógica e o **Template** é responsável pela apresentação dos dados.

**Vantagem principal:** O Django é “baterias incluídas”, o que significa que ele já traz várias funcionalidades prontas para uso, permitindo que os desenvolvedores se concentrem nas especificidades de seus projetos.

**1.2 Instalando o Python**

O primeiro passo para trabalhar com Django é garantir que você tenha o Python instalado. Django é escrito em Python, então você precisará do Python na sua máquina para começar.

**1.2.1 Verificando se o Python está instalado**

Abra o terminal (Windows PowerShell, CMD ou terminal do Linux/Mac) e execute o seguinte comando:

python --version

Se o Python já estiver instalado, o terminal exibirá a versão. Caso contrário, você precisará instalá-lo.

**1.2.2 Instalando o Python**

1. Acesse o site oficial do Python: [python.org](https://www.python.org).
2. Baixe a versão mais recente do Python.
3. Durante a instalação, certifique-se de marcar a opção “**Add Python to PATH**”.

Após a instalação, execute novamente o comando python --version para confirmar que tudo foi instalado corretamente.

**1.3 Configurando o Ambiente de Desenvolvimento**

**1.3.1 Escolhendo o Editor de Código**

Um bom editor de código é essencial para o desenvolvimento eficiente com Django. Recomendamos o **Visual Studio Code (VS Code)**, que oferece várias extensões úteis para Python e Django.

* Baixe e instale o VS Code: [VS Code](https://code.visualstudio.com/).
* Instale as extensões **Python** e **Django** no VS Code para facilitar o desenvolvimento.

**1.3.2 Criando um Projeto Django**

Agora que temos o Python instalado, vamos configurar nosso ambiente de desenvolvimento.

**1.3.2.1 Criando uma pasta para o projeto**

Escolha um local no seu sistema e crie uma nova pasta para o projeto:

mkdir learn\_log

cd learn\_log

**1.3.2.2 Criando um ambiente virtual**

No desenvolvimento Python, é uma boa prática criar ambientes virtuais para cada projeto. Isso garante que as dependências de um projeto não interfiram em outros.

Para criar um ambiente virtual, execute:

python -m venv ll\_env

Isso criará um ambiente virtual chamado ll\_env. Agora, você precisa ativá-lo.

**1.3.2.3 Ativando o ambiente virtual**

* No Windows:

.\ll\_env\Scripts\activate

* No macOS/Linux:

source ll\_env/bin/activate

Após ativar o ambiente virtual, você verá o nome dele entre parênteses no terminal, indicando que ele está ativo:

(ll\_env) $

**1.3.2.4 Instalando o Django**

Com o ambiente virtual ativo, podemos instalar o Django. Execute o seguinte comando:

pip install django

Após a instalação, verifique se tudo está funcionando corretamente:

django-admin --version

Isso deve retornar a versão do Django instalada.

**1.3.2.5 Criando o projeto Django**

Agora que o Django está instalado, vamos criar o projeto. Ainda no terminal, dentro da pasta do projeto, execute:

django-admin startproject learning\_log .

O ponto final (.) indica que queremos criar o projeto na pasta atual.

**1.4 Explorando a Estrutura de um Projeto Django**

Após a criação do projeto, você verá que vários arquivos e pastas foram gerados. Vamos entender o que cada um deles faz:

* **manage.py**: Um script útil para gerenciar o projeto. Com ele, podemos rodar o servidor, aplicar migrações, criar superusuários e muito mais.
* **learning\_log/**: A pasta do projeto principal que contém a configuração do Django.
  + **init.py**: Indica que a pasta é um pacote Python.
  + **settings.py**: Onde estão as configurações principais do projeto, como banco de dados, apps instalados e configurações de segurança.
  + **urls.py**: Define o roteamento da aplicação. É aqui que mapeamos URLs para funções.
  + **wsgi.py**: Usado para implantar o projeto em servidores web.
  + **asgi.py**: Configurações para servidores assíncronos (opcional).

**1.5 Rodando o Servidor de Desenvolvimento**

Com a estrutura do projeto criada, podemos rodar o servidor de desenvolvimento do Django. No terminal, dentro da pasta do projeto, execute:

python manage.py runserver

Se tudo estiver configurado corretamente, você verá a seguinte mensagem:

Starting development server at http://127.0.0.1:8000/

Abra um navegador e acesse http://127.0.0.1:8000/. Parabéns! Seu servidor Django está rodando.

**Capítulo 2: Criando Modelos e Trabalhando com Banco de Dados**

**2.1 O que é um Modelo no Django?**

Os **modelos** são a forma como o Django representa as tabelas do banco de dados usando Python. Em vez de escrever SQL diretamente, você define classes em Python que representam as tabelas e os campos do banco.

Cada modelo é uma classe Python que herda de models.Model, e cada atributo da classe representa uma coluna no banco de dados.

**2.1.1 Exemplo de um modelo básico**

Vamos criar nosso primeiro modelo. No terminal, execute o seguinte comando para criar um novo app chamado learnings:

python manage.py startapp learnings

Após isso, você verá que uma nova pasta learnings foi criada no projeto.

Agora, vamos definir um modelo chamado Topic para representar tópicos de aprendizado. Abra o arquivo models.py dentro da pasta learnings e adicione o seguinte código:

python

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

**2.1.2 Explicando o código**

* **models.Model**: O Topic herda de models.Model, que é a classe base para todos os modelos Django.
* **text**: Um campo de texto com um limite máximo de 200 caracteres.
* **date\_added**: Um campo de data e hora que automaticamente registra quando o tópico foi criado.
* **str()**: Retorna uma representação amigável do objeto quando ele for exibido (por exemplo, no admin do Django).

**2.1.3 Registrando o modelo no Admin**

Para visualizar e gerenciar nosso modelo no painel administrativo do Django, precisamos registrá-lo. Abra o arquivo admin.py dentro da pasta learnings e adicione o código abaixo:

python

from django.contrib import admin

from .models import Topic

admin.site.register(Topic)

**2.2 Criando e Aplicando Migrações**

No Django, as **migrações** são arquivos que mapeiam as mudanças nos modelos para o banco de dados. Sempre que você criar ou modificar um modelo, deverá criar uma migração para refletir essas alterações no banco de dados.

**2.2.1 Criando migrações**

No terminal, execute o seguinte comando para criar a migração do modelo Topic:

python manage.py makemigrations

Esse comando gera um arquivo de migração na pasta migrations do app learnings, que descreve as mudanças feitas no banco de dados.

**2.2.2 Aplicando migrações**

Para aplicar as migrações ao banco de dados, execute:

python manage.py migrate

Isso cria a tabela learnings\_topic no banco de dados.

**2.3 Explorando o Banco de Dados**

Agora que temos nossa tabela criada, podemos visualizar o banco de dados. Usaremos o **DB Browser for SQLite** (ou qualquer outro cliente de banco de dados SQLite) para abrir o arquivo db.sqlite3 que está na pasta raiz do projeto. Dentro desse arquivo, você verá a tabela learnings\_topic com os campos id, text, e date\_added.

**Capítulo 3: Trabalhando com Views e Templates**

**3.1 O que são Views no Django?**

As **views** em Django são funções ou classes que recebem uma requisição HTTP, processam os dados (se necessário) e retornam uma resposta. Elas são responsáveis pela lógica de negócios do aplicativo e frequentemente renderizam templates para exibir conteúdo dinâmico nas páginas.

Há dois tipos de views no Django:

1. **Views baseadas em funções (FBV - Function Based Views)**: Funções que processam uma requisição e retornam uma resposta.
2. **Views baseadas em classes (CBV - Class Based Views)**: Classes que fornecem uma maneira mais estruturada e reutilizável de criar views.

Neste capítulo, vamos focar nas **views baseadas em funções**, que são ideais para iniciantes, mas também mencionaremos CBVs mais tarde.

**3.2 Criando uma View Simples**

Vamos começar criando uma view básica que exibirá a página inicial da nossa aplicação.

**3.2.1 Definindo a View**

Abra o arquivo views.py dentro da pasta do app learnings e adicione o seguinte código:

python

from django.shortcuts import render

def index(request):

"""A página inicial da aplicação Learning Log."""

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

**Explicação:**

* A função index() recebe um objeto request (que contém dados sobre a requisição HTTP) e usa o método render() para renderizar um template chamado index.html.
* O método render() combina o template com um contexto (dados a serem exibidos) e retorna uma resposta HTTP.

**3.2.2 Mapeando a URL para a View**

Agora que temos nossa view, precisamos mapeá-la para uma URL específica. Isso é feito no arquivo urls.py.

1. Crie um novo arquivo urls.py dentro da pasta do app learnings e adicione o seguinte código:

python

from django.urls import path

from . import views

urlpatterns = [

# Página inicial

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

]

1. No arquivo urls.py do projeto (que está na pasta learning\_log), modifique-o para incluir o arquivo de URLs do app learnings:

python

from django.urls import path, include

urlpatterns = [

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

path('', include('learnings.urls')),

]

**Explicação:**

* Estamos dizendo ao Django para procurar as rotas do app learnings quando a URL raiz ('/') for acessada.
* O arquivo learnings/urls.py define que a URL vazia ('') chama a função index() da view.

**3.3 Trabalhando com Templates**

O **template** é um arquivo HTML que define a estrutura e o layout de uma página web. Django utiliza o sistema de templates para gerar páginas dinâmicas.

**3.3.1 Criando o Template**

Agora vamos criar o template index.html para a nossa página inicial.

1. Dentro da pasta learnings, crie uma nova pasta chamada templates.
2. Dentro da pasta templates, crie outra pasta chamada learnings (Django espera que os templates estejam organizados dessa forma, com uma pasta que leva o nome do app).
3. Crie o arquivo index.html dentro da pasta learnings/templates/learnings e adicione o seguinte conteúdo:

html

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Learning Log</title>

</head>

<body>

<h1>Learning Log</h1>

<p>Bem-vindo ao Learning Log, uma aplicação para organizar seus tópicos de aprendizado.</p>

</body>

</html>

Agora, quando você acessar a URL http://127.0.0.1:8000/, verá a página inicial com a mensagem de boas-vindas.

**3.3.2 Explicando a Renderização do Template**

A função render() na view associa o template index.html ao objeto request, gerando uma página HTML para ser exibida no navegador.

* **render(request, 'learnings/index.html')**: Isso combina o template index.html com um "contexto" (se houver) e retorna a página para o navegador.
* A separação de templates e lógica no Django permite que os desenvolvedores criem uma interface clara entre a lógica da aplicação e a apresentação dos dados.

**3.4 Introdução ao Contexto e Dados Dinâmicos**

Em muitas aplicações web, as páginas não são estáticas. Elas frequentemente exibem dados dinâmicos vindos de um banco de dados. No Django, isso é feito passando um **contexto** da view para o template.

**3.4.1 Exemplo: Passando Dados para o Template**

Vamos modificar nossa view para passar dados dinâmicos para o template.

No arquivo views.py, modifique a função index() para incluir um contexto:

python

def index(request):

"""A página inicial da aplicação Learning Log."""

contexto = {'mensagem': 'Esta é uma mensagem dinâmica vinda da view!'}

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

No arquivo index.html, modifique o conteúdo para exibir essa mensagem:

html

<h1>Learning Log</h1>

<p>{{ mensagem }}</p>

**Explicação:**

* O dicionário contexto contém dados que serão passados para o template. A chave do dicionário (mensagem) pode ser usada no template com a sintaxe {{ mensagem }}.
* O resultado será uma página que exibe a mensagem dinâmica.

**Capítulo 4: Trabalhando com Modelos e Bancos de Dados**

**4.1 Revisão de Modelos e ORM do Django**

Como vimos no capítulo anterior, o **ORM (Object-Relational Mapping)** do Django permite que você interaja com o banco de dados usando classes Python (os modelos). No capítulo 2, criamos um modelo básico chamado Topic. Agora vamos explorar o relacionamento entre modelos e como gerenciar esses dados no painel administrativo e no código.

**4.2 Definindo Relacionamentos entre Modelos**

**4.2.1 Criando um Modelo de Relacionamento**

Vamos criar um novo modelo chamado Entry, que representará as **anotações** relacionadas a um Topic. Cada Entry estará relacionado a um Topic através de uma chave estrangeira.

Abra o arquivo models.py dentro da pasta learnings e adicione o seguinte código para definir o modelo Entry:

python

from django.db import models

class Entry(models.Model):

"""Anotações relacionadas a um tópico específico."""

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 string com os primeiros 50 caracteres da entrada."""

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

**4.2.2 Explicando o Código**

* **ForeignKey**: A chave estrangeira (ForeignKey) cria um relacionamento entre o modelo Entry e o modelo Topic. O argumento on\_delete=models.CASCADE indica que, se um Topic for deletado, todas as Entry relacionadas a ele também serão deletadas.
* **TextField**: Um campo de texto sem limite de caracteres.
* **Meta class**: A opção verbose\_name\_plural ajusta o plural exibido no painel administrativo para “entries” em vez de “Entrys”.
* **str()**: A função \_\_str\_\_() retorna uma string com os primeiros 50 caracteres do campo text, útil para exibir uma prévia das anotações.

**4.3 Registrando os Modelos no Admin**

Assim como fizemos com o modelo Topic, precisamos registrar o modelo Entry no painel administrativo.

Abra o arquivo admin.py e adicione o seguinte código:

python

from .models import Entry

admin.site.register(Entry)

Isso permitirá que você gerencie as entradas diretamente no painel administrativo.

**4.4 Criando Migrations e Aplicando-as**

Sempre que você criar ou modificar um modelo, é necessário criar e aplicar migrações para atualizar o banco de dados.

1. **Criando migrações**:

python manage.py makemigrations

1. **Aplicando migrações**:

python manage.py migrate

Após aplicar as migrações, o modelo Entry será refletido no banco de dados.

**4.5 Trabalhando com o Django Shell**

O **Django Shell** é uma ferramenta muito útil para testar modelos e manipular dados diretamente no banco de dados. Para acessá-lo, execute o seguinte comando no terminal:

python manage.py shell

**4.5.1 Exemplo: Inserindo e Recuperando Dados no Shell**

Vamos criar um novo Topic e adicionar algumas Entry relacionadas diretamente pelo shell.

1. **Importe os modelos**:

python

from learnings.models import Topic, Entry

1. **Crie um novo Topic**:

python

novo\_topico = Topic(text='Estudando Django')

novo\_topico.save()

1. **Adicione uma Entry ao Topic**:

python

entrada = Entry(topic=novo\_topico, text='Aprendendo como criar modelos no Django.')

entrada.save()

1. **Recupere todas as Entry relacionadas ao Topic**:

python

entries = novo\_topico.entry\_set.all()

for entry in entries:

print(entry.text)

**Capítulo 5: Painel Administrativo e Gerenciamento de Dados**

**5.1 Explorando o Painel Administrativo**

O painel administrativo do Django é uma ferramenta extremamente poderosa para gerenciar os dados da aplicação. Ele é gerado automaticamente com base nos modelos que você registra. Vamos explorar como personalizar o painel para torná-lo mais útil.

**5.1.1 Criando um Superusuário**

Para acessar o painel administrativo, primeiro precisamos criar um **superusuário**, que é um usuário com permissões de administrador.

No terminal, execute o seguinte comando:

python manage.py createsuperuser

Você será solicitado a inserir um nome de usuário, e-mail e senha. Após a criação, você poderá acessar o painel administrativo em http://127.0.0.1:8000/admin.

**5.1.2 Personalizando o Painel Administrativo**

Além de registrar os modelos, podemos personalizar o que aparece no painel. Por exemplo, podemos exibir mais campos na lista de objetos e criar filtros de busca.

Abra o arquivo admin.py e modifique o código para personalizar a exibição de Topic e Entry no painel:

python

class TopicAdmin(admin.ModelAdmin):

list\_display = ('text', 'date\_added')

search\_fields = ['text']

class EntryAdmin(admin.ModelAdmin):

list\_display = ('text', 'topic', 'date\_added')

list\_filter = ['date\_added', 'topic']

admin.site.register(Topic, TopicAdmin)

admin.site.register(Entry, EntryAdmin)

**5.1.3 Explicação da Personalização**

* **list\_display**: Define quais campos serão exibidos na lista de objetos no painel administrativo.
* **search\_fields**: Permite realizar buscas por palavras-chave nos campos especificados.
* **list\_filter**: Adiciona filtros laterais para facilitar a navegação e segmentação dos registros.

**5.2 Gerenciando Dados no Painel Administrativo**

Agora que nosso painel está personalizado, vamos explorar como criar, editar e deletar registros diretamente no painel. Acesse o painel em http://127.0.0.1:8000/admin e navegue até os modelos Topic e Entry.

* **Adicionando um novo Topic**: Preencha o formulário com o título do tópico.
* **Adicionando uma nova Entry**: Selecione o Topic relacionado e insira o texto da anotação.

Com esses capítulos, os alunos serão capazes de criar, gerenciar e personalizar suas aplicações Django, com entendimento de como views, templates e modelos interagem, além de explorar o potencial do painel administrativo do Django para gerenciar dados.

**Capítulo 6: Criando Rotas e Páginas Dinâmicas**

**6.1 O que são Rotas no Django?**

As **rotas** são a maneira como o Django determina qual código deve ser executado quando um usuário acessa uma URL específica. Cada rota está associada a uma view que processa a requisição e retorna a resposta correspondente.

**6.1.1 Estrutura de um arquivo de URLs**

No Django, as rotas são definidas em arquivos urls.py. Cada app pode ter seu próprio arquivo urls.py, onde as rotas são mapeadas para as views apropriadas.

**6.1.2 Criando uma nova rota para Tópicos**

Vamos adicionar uma nova rota que exibirá todos os tópicos cadastrados. Abra o arquivo urls.py dentro da pasta do app learnings e adicione o seguinte código:

python

from django.urls import path

from . import views

urlpatterns = [

path('', views.index, name='index'), # Página inicial

path('topics/', views.topics, name='topics'), # Página de Tópicos

]

**6.2 Criando a View para Exibir Tópicos**

Agora, vamos criar a view topics() que irá buscar todos os tópicos do banco de dados e exibi-los em um template.

**6.2.1 Definindo a View**

No arquivo views.py, adicione a função topics():

python

def topics(request):

"""Exibe todos os tópicos cadastrados."""

all\_topics = Topic.objects.all() # Busca todos os tópicos

contexto = {'topics': all\_topics} # Cria um contexto com os tópicos

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

**Explicação:**

* A função topics() usa Topic.objects.all() para buscar todos os objetos do modelo Topic e armazena-os na variável all\_topics.
* Um dicionário de contexto é criado com a chave topics, que é passado para o template topics.html.

**6.3 Criando o Template para a Página de Tópicos**

Agora precisamos criar o template topics.html para exibir a lista de tópicos.

1. Crie o arquivo topics.html dentro da pasta learnings/templates/learnings e adicione o seguinte conteúdo:

html

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Tópicos - Learning Log</title>

</head>

<body>

<h1>Tópicos</h1>

<ul>

{% for topic in topics %}

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

{% empty %}

<li>Não há tópicos cadastrados.</li>

{% endfor %}

</ul>

<a href="{% url 'index' %}">Voltar para a página inicial</a>

</body>

</html>

**6.3.1 Explicação do Template**

* O loop {% for topic in topics %} itera sobre cada tópico passado no contexto e exibe o texto do tópico em uma lista não ordenada.
* A cláusula {% empty %} exibe uma mensagem caso não haja tópicos cadastrados.
* O link no final permite que o usuário volte para a página inicial, utilizando a tag {% url 'index' %} para gerar a URL corretamente.

**6.4 Testando a Página de Tópicos**

Agora que a rota e a view estão configuradas, inicie o servidor Django e acesse http://127.0.0.1:8000/topics/ para visualizar a lista de tópicos cadastrados. Você deve ver os tópicos que criou no painel administrativo.

**Capítulo 7: Interagindo com Formulários**

**7.1 Por que usar Formulários no Django?**

Os **formulários** permitem que os usuários enviem dados para o servidor. No Django, os formulários oferecem várias funcionalidades, como validação automática e integração com modelos, facilitando a manipulação de dados.

**7.1.1 Criando um Formulário para Tópicos**

Vamos criar um formulário que permitirá aos usuários adicionar novos tópicos à aplicação.

**7.1.2 Definindo o Formulário**

No app learnings, crie um novo arquivo chamado forms.py e adicione o seguinte código:

python

from django import forms

from .models import Topic

class TopicForm(forms.ModelForm):

"""Formulário para adicionar novos tópicos."""

class Meta:

model = Topic

fields = ['text']

labels = {'text': ''} # Remove o rótulo padrão

**Explicação:**

* O TopicForm é uma subclasse de forms.ModelForm, que é um formulário baseado no modelo Topic.
* fields especifica quais campos do modelo serão incluídos no formulário.
* labels permite customizar ou remover os rótulos padrão dos campos.

**7.2 Criando a View para Processar o Formulário**

Agora que temos o formulário, precisamos criar uma view que permitirá adicionar novos tópicos.

**7.2.1 Definindo a View**

No arquivo views.py, adicione a seguinte função:

python

def new\_topic(request):

"""Adiciona um novo tópico."""

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() # Salva o novo tópico no banco de dados

return redirect('topics') # Redireciona para a página de tópicos

contexto = {'form': form} # Cria um contexto com o formulário

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

**Explicação:**

* A view new\_topic() verifica se a requisição é do tipo POST. Se não for, um formulário vazio é criado.
* Se a requisição for POST, o Django tenta validar e salvar o formulário. Se o formulário for válido, o novo tópico é salvo e o usuário é redirecionado para a página de tópicos.

**7.3 Mapeando a Nova Rota**

Agora, precisamos mapear a nova view para uma URL. No arquivo urls.py, adicione a seguinte linha:

python

path('new\_topic/', views.new\_topic, name='new\_topic'), # Página para adicionar um novo tópico

**7.4 Criando o Template para o Formulário**

Agora precisamos criar um template para o formulário. Crie o arquivo new\_topic.html dentro da pasta learnings/templates/learnings e adicione o seguinte conteúdo:

html

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Novo Tópico - Learning Log</title>

</head>

<body>

<h1>Novo Tópico</h1>

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

{% csrf\_token %}

{{ form.as\_p }} <!-- Renderiza o formulário -->

<button type="submit">Adicionar Tópico</button>

</form>

<a href="{% url 'topics' %}">Voltar para a lista de tópicos</a>

</body>

</html>

**7.4.1 Explicação do Template**

* O formulário usa a tag {% csrf\_token %} para proteção contra ataques CSRF.
* {{ form.as\_p }} renderiza os campos do formulário dentro de tags <p>, formatando-os de forma simples.
* O botão "Adicionar Tópico" permite que o usuário envie o formulário.

**7.5 Testando o Formulário**

Inicie o servidor e acesse http://127.0.0.1:8000/new\_topic/ para adicionar um novo tópico. Preencha o formulário e clique em "Adicionar Tópico". Após o envio, você deve ser redirecionado para a página de tópicos, onde o novo tópico deve aparecer na lista.

**Capítulo 8: Estilizando a Aplicação com CSS e Herança de Templates**

**8.1 Introdução ao CSS no Django**

O CSS (Cascading Style Sheets) é usado para estilizar o layout e a aparência das páginas web. O Django facilita a inclusão de arquivos CSS em seus templates, permitindo que você crie interfaces atraentes e responsivas.

**8.1.1 Estrutura de Pastas para Arquivos Estáticos**

Para incluir arquivos CSS, precisamos criar uma estrutura de pastas para arquivos estáticos:

1. Dentro da pasta learnings, crie uma nova pasta chamada static.
2. Dentro da pasta static, crie outra pasta chamada learnings.

O caminho final deve ser learnings/static/learnings/. É aqui que você colocará seus arquivos CSS.

**8.1.2 Criando um Arquivo CSS**

Crie um arquivo chamado styles.css dentro da pasta learnings/static/learnings e adicione o seguinte conteúdo:

css

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

margin: 0;

padding: 20px;

}

h1 {

color: #333;

}

ul {

list-style-type: none;

padding: 0;

}

li {

background: #fff;

margin: 5px 0;

padding: 10px;

border: 1px solid #ddd;

}

**Explicação:**

* O CSS acima estiliza o corpo da página, os títulos e as listas, melhorando a apresentação visual.

**8.2 Incluindo o CSS nos Templates**

Para que o Django possa encontrar e usar o arquivo CSS, precisamos incluir um bloco que carrega os arquivos estáticos em nossos templates.

**8.2.1 Atualizando o Template Base**

Crie um arquivo chamado base.html dentro da pasta learnings/templates/learnings e adicione o seguinte conteúdo:

html

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>{% block title %}Learning Log{% endblock %}</title>

<link rel="stylesheet" href="{% static 'learnings/styles.css' %}">

</head>

<body>

<header>

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

<nav>

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

<a href="{% url 'new\_topic' %}">Novo Tópico</a>

</nav>

</header>

<main>

{% block content %}

{% endblock %}

</main>

</body>

</html>

**Explicação:**

* O bloco {% block title %} permite que as páginas que herdam desse template definam um título personalizado.
* A linha <link rel="stylesheet" href="{% static 'learnings/styles.css' %}"> inclui o arquivo CSS no template, permitindo que suas regras de estilo sejam aplicadas.

**8.3 Usando Herança de Templates**

Agora que temos o template base, vamos modificar nossos outros templates para que herdem do base.html.

**8.3.1 Atualizando o Template index.html**

Altere o index.html para o seguinte:

html

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

{% block content %}

<p>Bem-vindo ao Learning Log, uma aplicação para organizar seus tópicos de aprendizado.</p>

{% endblock %}

**8.3.2 Atualizando o Template topics.html**

Faça o mesmo com o topics.html:

html

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

{% block content %}

<h1>Tópicos</h1>

<ul>

{% for topic in topics %}

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

{% empty %}

<li>Não há tópicos cadastrados.</li>

{% endfor %}

</ul>

<a href="{% url 'new\_topic' %}">Adicionar novo tópico</a>

{% endblock %}

**8.4 Testando a Estilização**

Agora inicie o servidor e acesse http://127.0.0.1:8000/ e http://127.0.0.1:8000/topics/. Você deve ver que os estilos do CSS foram aplicados, tornando as páginas mais agradáveis visualmente.

Com esses capítulos, revisamos como criar rotas, trabalhar com formulários, adicionar estilos e utilizar herança de templates para construir uma aplicação Django organizada e esteticamente agradável.

**Capítulo 9: Criando Páginas Dinâmicas com Anotações**

**9.1 Introdução às Anotações**

No nosso projeto, as **anotações** (ou **entries**) são informações que os usuários podem associar a um tópico específico. Cada entrada deve conter texto e estar vinculada a um tópico. Neste capítulo, aprenderemos a criar páginas dinâmicas que exibem as entradas de um tópico específico.

**9.1.1 Estruturando as Rotas para Anotações**

Vamos adicionar uma nova rota que permitirá visualizar as entradas de um tópico específico. Primeiro, modifique o arquivo urls.py do app learnings para incluir a nova rota:

python

path('topics/<int:topic\_id>/', views.topic, name='topic'), # Página de um tópico específico

Essa rota captura um número inteiro na URL, que representa o ID do tópico.

**9.2 Criando a View para um Tópico Específico**

Agora vamos criar a view topic() que exibirá um único tópico junto com suas entradas.

**9.2.1 Definindo a View**

No arquivo views.py, adicione a seguinte função:

python

def topic(request, topic\_id):

"""Exibe um tópico específico e suas entradas associadas."""

topic = Topic.objects.get(id=topic\_id) # Busca o tópico pelo ID

entries = topic.entry\_set.order\_by('-date\_added') # Obtém entradas relacionadas ao tópico

contexto = {'topic': topic, 'entries': entries} # Cria um contexto

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

**Explicação:**

* A função topic() recebe o ID do tópico como um argumento.
* Usamos Topic.objects.get(id=topic\_id) para buscar o tópico correspondente.
* O método entry\_set é usado para acessar todas as entradas associadas a esse tópico, ordenadas por data.

**9.3 Criando o Template para a Página do Tópico**

Agora precisamos criar o template topic.html para exibir o tópico e suas entradas. Crie o arquivo topic.html na pasta learnings/templates/learnings e adicione o seguinte conteúdo:

html

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

{% block content %}

<h2>{{ topic.text }}</h2>

<p>Entradas:</p>

<ul>

{% for entry in entries %}

<li>

<p>{{ entry.date\_added|date:"d M Y H:i" }}</p> <!-- Formata a data -->

<p>{{ entry.text|linebreaks }}</p> <!-- Adiciona quebras de linha -->

</li>

{% empty %}

<li>Não há entradas para este tópico.</li>

{% endfor %}

</ul>

<a href="{% url 'new\_entry' topic.id %}">Adicionar nova entrada</a> <!-- Link para adicionar entradas -->

{% endblock %}

**9.3.1 Explicação do Template**

* **{{ topic.text }}** exibe o título do tópico.
* O loop {% for entry in entries %} itera sobre todas as entradas do tópico e exibe a data e o texto.
* O filtro date:"d M Y H:i" formata a data de forma amigável, e o filtro linebreaks insere quebras de linha no texto da entrada.
* Um link é fornecido para adicionar uma nova entrada ao tópico.

**9.4 Criando a View para Adicionar Entradas**

Agora, vamos criar uma view para permitir que os usuários adicionem novas entradas a um tópico.

**9.4.1 Definindo a View**

No arquivo views.py, adicione a seguinte função:

python

def new\_entry(request, topic\_id):

"""Adiciona uma nova entrada a um tópico específico."""

topic = Topic.objects.get(id=topic\_id) # Obtém o tópico

if request.method != 'POST':

form = EntryForm() # Cria um formulário vazio

else:

form = EntryForm(request.POST) # Cria um formulário com os dados do POST

if form.is\_valid():

new\_entry = form.save(commit=False) # Não salva imediatamente

new\_entry.topic = topic # Associa a entrada ao tópico

new\_entry.save() # Salva a nova entrada no banco de dados

return redirect('topic', topic\_id=topic.id) # Redireciona para a página do tópico

contexto = {'form': form, 'topic': topic} # Cria um contexto

return render(request, 'learnings/new\_entry.html', contexto)

**9.4.2 Explicação da View**

* A view new\_entry() começa obtendo o tópico correspondente ao topic\_id.
* Se a requisição não for do tipo POST, um formulário vazio é criado. Caso contrário, ele tenta validar o formulário com os dados enviados.
* O método commit=False permite que você modifique o objeto antes de salvá-lo (neste caso, associando-o ao tópico).

**9.5 Mapeando a Nova Rota para Entradas**

No arquivo urls.py, adicione a nova rota:

python

path('topics/<int:topic\_id>/new\_entry/', views.new\_entry, name='new\_entry'), # Adiciona uma nova entrada ao tópico

**9.6 Criando o Formulário para Entradas**

Para adicionar uma entrada, precisamos de um formulário. Crie um novo arquivo chamado forms.py (se ainda não existir) e adicione o seguinte código:

python

from django import forms

from .models import Entry

class EntryForm(forms.ModelForm):

"""Formulário para adicionar novas entradas."""

class Meta:

model = Entry

fields = ['text'] # Campo de texto

labels = {'text': ''} # Remove o rótulo padrão

**9.7 Criando o Template para Adicionar Entradas**

Crie o arquivo new\_entry.html na pasta learnings/templates/learnings e adicione o seguinte conteúdo:

html

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

{% block content %}

<h2>Adicionar nova entrada para "{{ topic.text }}"</h2>

<form action="{% url 'new\_entry' topic.id %}" method="post">

{% csrf\_token %}

{{ form.as\_p }} <!-- Renderiza o formulário -->

<button type="submit">Adicionar Entrada</button>

</form>

<a href="{% url 'topic' topic.id %}">Voltar para o tópico</a>

{% endblock %}

**Explicação:**

* O template renderiza o formulário e permite que o usuário adicione uma nova entrada ao tópico.
* O link permite retornar à página do tópico.

**9.8 Testando a Funcionalidade de Anotações**

Inicie o servidor e acesse http://127.0.0.1:8000/topics/1/new\_entry/ (substitua 1 pelo ID do tópico desejado) para adicionar uma nova entrada. Após adicionar, você deve ser redirecionado de volta para a página do tópico, onde a nova entrada deve aparecer.

**Capítulo 10: Aprimorando a Navegação e a Experiência do Usuário**

**10.1 Melhorando a Navegação no Site**

Uma boa navegação é fundamental para melhorar a experiência do usuário. Neste capítulo, vamos adicionar links de navegação para tornar o site mais intuitivo.

**10.1.1 Atualizando o Template Base**

No arquivo base.html, atualize a seção do cabeçalho para incluir links para as páginas de tópicos e nova entrada:

html

<header>

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

<nav>

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

<a href="{% url 'new\_topic' %}">Novo Tópico</a>

</nav>

</header>

**Explicação:**

* Os links foram adicionados para permitir que os usuários naveguem facilmente entre as páginas principais da aplicação.

**10.2 Adicionando Funcionalidades de Edição e Exclusão**

Agora que temos as páginas de tópicos e entradas, vamos adicionar a capacidade de editar e excluir tópicos e entradas.

**10.2.1 Criando a View para Editar um Tópico**

No arquivo views.py, adicione a seguinte função:

python

def edit\_topic(request, topic\_id):

"""Edita um tópico existente."""

topic = Topic.objects.get(id=topic\_id) # Obtém o tópico

if request.method != 'POST':

form = TopicForm(instance=topic) # Cria um formulário com os dados do tópico

else:

form = TopicForm(request.POST, instance=topic) # Atualiza o formulário com os dados do POST

if form.is\_valid():

form.save() # Salva as alterações

return redirect('topics') # Redireciona para a página de tópicos

contexto = {'form': form, 'topic': topic} # Cria um contexto

return render(request, 'learnings/edit\_topic.html', contexto)

**10.2.2 Mapeando a Rota de Edição**

No arquivo urls.py, adicione a nova rota:

python

path('topics/<int:topic\_id>/edit/', views.edit\_topic, name='edit\_topic'), # Rota para editar um tópico

**10.2.3 Criando o Template de Edição**

Crie o arquivo edit\_topic.html na pasta learnings/templates/learnings e adicione o seguinte conteúdo:

html

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

{% block content %}

<h2>Editando Tópico: "{{ topic.text }}"</h2>

<form action="{% url 'edit\_topic' topic.id %}" method="post">

{% csrf\_token %}

{{ form.as\_p }} <!-- Renderiza o formulário -->

<button type="submit">Salvar Alterações</button>

</form>

<a href="{% url 'topic' topic.id %}">Voltar para o tópico</a>

{% endblock %}

**10.3 Criando a View para Excluir um Tópico**

Vamos agora adicionar a funcionalidade para excluir um tópico.

**10.3.1 Definindo a View de Exclusão**

No arquivo views.py, adicione a seguinte função:

python

def delete\_topic(request, topic\_id):

"""Exclui um tópico existente."""

topic = Topic.objects.get(id=topic\_id) # Obtém o tópico

if request.method == 'POST':

topic.delete() # Exclui o tópico

return redirect('topics') # Redireciona para a página de tópicos

contexto = {'topic': topic} # Cria um contexto

return render(request, 'learnings/delete\_topic.html', contexto)

**10.3.2 Mapeando a Rota de Exclusão**

Adicione a rota de exclusão ao arquivo urls.py:

python

path('topics/<int:topic\_id>/delete/', views.delete\_topic, name='delete\_topic'), # Rota para excluir um tópico

**10.3.3 Criando o Template de Exclusão**

Crie o arquivo delete\_topic.html na pasta learnings/templates/learnings e adicione o seguinte conteúdo:

html

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

{% block content %}

<h2>Excluir Tópico: "{{ topic.text }}"</h2>

<p>Tem certeza de que deseja excluir este tópico?</p>

<form action="{% url 'delete\_topic' topic.id %}" method="post">

{% csrf\_token %}

<button type="submit">Confirmar Exclusão</button>

</form>

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

{% endblock %}

**10.4 Testando as Funcionalidades de Edição e Exclusão**

Agora inicie o servidor e teste as seguintes funcionalidades:

* Acesse a página de tópicos, clique no tópico e use o link de edição para modificar o tópico.
* Utilize o link de exclusão para remover o tópico.

Verifique se as alterações estão refletindo corretamente nas páginas.

**Capítulo 11: Adicionando Validação e Tratamento de Erros**

**11.1 A Importância da Validação de Dados**

A validação de dados é crucial para garantir que os dados inseridos pelos usuários sejam corretos e estejam em conformidade com as regras da aplicação. Neste capítulo, aprenderemos a implementar validações nos formulários e a lidar com erros de maneira eficaz.

**11.1.1 Validando os Dados do Formulário**

No arquivo forms.py, podemos adicionar validações personalizadas ao formulário TopicForm:

python

from django.core.exceptions import ValidationError

class TopicForm(forms.ModelForm):

"""Formulário para adicionar novos tópicos."""

class Meta:

model = Topic

fields = ['text']

labels = {'text': ''}

def clean\_text(self):

"""Valida o campo de texto para não aceitar entradas vazias."""

text = self.cleaned\_data.get('text')

if not text:

raise ValidationError('O campo de texto não pode estar vazio.')

return text

**11.1.2 Explicação da Validação**

* O método clean\_text() é chamado automaticamente pelo Django durante a validação do formulário.
* self.cleaned\_data contém os dados limpos do formulário. Se o campo text estiver vazio, uma exceção ValidationError é lançada, o que impede que o formulário seja salvo.

**11.2 Exibindo Erros no Template**

Agora que temos validações, precisamos exibir mensagens de erro nos templates.

**11.2.1 Atualizando o Template de Novo Tópico**

No arquivo new\_topic.html, adicione a lógica para exibir mensagens de erro:

html

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

{% block content %}

<h2>Novo Tópico</h2>

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

{% csrf\_token %}

{{ form.non\_field\_errors }} <!-- Exibe erros que não estão associados a um campo específico -->

{{ form.as\_p }} <!-- Renderiza o formulário -->

<button type="submit">Adicionar Tópico</button>

</form>

<a href="{% url 'topics' %}">Voltar para a lista de tópicos</a>

{% endblock %}

**11.2.2 Atualizando o Template de Edição**

Faça o mesmo no arquivo edit\_topic.html para exibir erros durante a edição de tópicos.

**11.3 Tratando Erros na View**

Vamos garantir que os erros sejam tratados corretamente na view. No arquivo views.py, atualize a função new\_topic() para incluir a lógica de erro:

python

def new\_topic(request):

"""Adiciona um novo tópico."""

if request.method != 'POST':

form = TopicForm() # Cria um formulário vazio

else:

form = TopicForm(request.POST) # Cria um formulário com os dados do POST

if form.is\_valid():

form.save() # Salva o novo tópico no banco de dados

return redirect('topics') # Redireciona para a página de tópicos

else:

# Caso o formulário não seja válido, os erros serão exibidos no template

contexto = {'form': form}

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

**11.4 Testando a Validação e Tratamento de Erros**

Inicie o servidor e teste as funcionalidades:

* Tente adicionar um tópico vazio e veja se a mensagem de erro é exibida corretamente.
* Verifique se os erros são mostrados na página de adição e edição de tópicos.

Com esses capítulos, vocês aprenderão a trabalhar com entradas dinâmicas, aprimorar a navegação na aplicação, implementar a edição e exclusão de dados, além de tratar erros e validar entradas de forma eficaz.

Esses conceitos são fundamentais para construir aplicações web mais robustas e seguras.