Desenvolvendo um CRUD com Django

Introdução

- **O que é Django?**
  - Framework web baseado em Python
  - Principais características e vantagens
- **O que é CRUD?**
  - Definição de CRUD (Create, Read, Update, Delete)
  - Importância de CRUDs em aplicações

Capítulo 1: Configuração do Ambiente

- **Instalando o Django**
  - Requisitos do sistema
  - Instalando o Django via pip
- **Criando um Projeto Django**
  - Estrutura básica do projeto
  - Configurações iniciais

Capítulo 2: Criando a Aplicação

- **Iniciando uma Aplicação Django**
  - Comando para criar uma aplicação
  - Integrando a aplicação ao projeto
- **Definindo o Modelo**
  - Criando modelos com Django ORM
  - Migrações e aplicação de migrações

Capítulo 3: Configurando o Admin

- **Configurando o Django Admin**
  - Registro de modelos no admin
  - Customização básica do admin

Capítulo 4: Criando as Vistas e URLs

- **Definindo URLs**
  - Configuração de URLs no Django
  - Incluindo URLs da aplicação no projeto
- **Criando Vistas**
  - Tipos de vistas (funções e classes)
  - Vistas para operações de CRUD

Capítulo 5: Desenvolvendo os Templates

- **Introdução aos Templates Django**
    - Sistema de templates do Django
    - Criando e organizando templates
- **Templates para CRUD**
    - Template para criação de objetos
    - Template para leitura (listagem e detalhes)
    - Template para atualização de objetos
    - Template para exclusão de objetos

Capítulo 6: Implementando Funcionalidades de CRUD

- **Operação de Criação**
    - Formulários Django
    - Vistas e templates para criação
- **Operação de Leitura**
    - Listando objetos
    - Exibindo detalhes de um objeto
- **Operação de Atualização**
    - Formulários para edição
    - Vistas e templates para atualização
- **Operação de Exclusão**
    - Confirmando a exclusão
    - Vistas e templates para exclusão

Capítulo 7: Melhorando a Aplicação

- **Validação de Formulários**
    - Validação no Django
    - Mensagens de erro e feedback para o usuário
- **Autenticação e Autorização**
    - Sistema de autenticação do Django
    - Restringindo acesso a operações de CRUD

Capítulo 8: Testando a Aplicação

- **Testes Unitários**
    - Criando testes para modelos
    - Testando vistas e templates
- **Testes de Integração**
    - Testando o fluxo completo do CRUD

Conclusão

- **Resumo do Desenvolvimento**
    - Recapitulando os passos

        - Dicas e melhores práticas
- **Próximos Passos**
    - Recursos adicionais para aprendizado
    - Sugestões de projetos para prática

Introdução

O que é Django?

Django é um framework web de alto nível baseado na linguagem de programação Python. Ele foi projetado para simplificar o desenvolvimento de aplicações web, permitindo que desenvolvedores construam projetos rapidamente e de forma eficiente. Entre suas principais características estão o mapeamento objeto-relacional (ORM), um sistema de templates poderoso, um sistema de roteamento de URLs flexível, e um sistema de administração automática.

O que é CRUD?

CRUD é um acrônimo que representa as quatro operações básicas de persistência de dados: Create (Criar), Read (Ler), Update (Atualizar), e Delete (Excluir). Em qualquer aplicação web que interaja com um banco de dados, a implementação dessas operações é essencial para gerenciar os dados.

Capítulo 1: Configuração do Ambiente

Instalando o Django

Para começar a desenvolver com Django, é necessário ter Python instalado no seu sistema. Você pode instalar o Django usando o gerenciador de pacotes `pip`.

```
bash
Copiar código
# Instalando Django
pip install django
```

Criando um Projeto Django

Depois de instalar o Django, você pode criar um novo projeto usando o comando `django-admin startproject`. Este comando cria a estrutura básica de diretórios e arquivos necessários para um projeto Django.

```
bash
Copiar código
# Criando um projeto Django
django-admin startproject meu_projeto

# Entrando no diretório do projeto
cd meu_projeto

# Iniciando o servidor de desenvolvimento
python manage.py runserver
```

O comando `runserver` inicia o servidor de desenvolvimento do Django, permitindo que você veja sua aplicação em funcionamento no navegador.

Capítulo 2: Criando a Aplicação

Iniciando uma Aplicação Django

Dentro do seu projeto Django, você pode criar várias aplicações. Cada aplicação é um módulo independente que pode ser reutilizado em diferentes projetos. Para criar uma nova aplicação, use o comando `startapp`.

```
bash
Copiar código
# Criando uma aplicação chamada 'app_crud'
python manage.py startapp app_crud
```

Integrando a Aplicação ao Projeto

Após criar a aplicação, você deve registrá-la no arquivo `settings.py` do seu projeto para que o Django a reconheça.

```
python
Copiar código
# Adicionando a aplicação ao settings.py
INSTALLED_APPS = [
    ...
    'app_crud',
]
```

Definindo o Modelo

No Django, os modelos são a forma de definir a estrutura dos dados. Você define seus modelos no arquivo `models.py` da sua aplicação. Vamos criar um modelo chamado `Item`.

```
python
Copiar código
# models.py em app_crud
from django.db import models

class Item(models.Model):
    nome = models.CharField(max_length=100)
    descricao = models.TextField()
    preco = models.DecimalField(max_digits=10, decimal_places=2)
    criado_em = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.nome
```

Após definir o modelo, você precisa criar e aplicar as migrações para que o Django crie a tabela correspondente no banco de dados.

```
bash
Copiar código
# Criando migrações
python manage.py makemigrations

# Aplicando migrações
```

```
python manage.py migrate
```

Capítulo 3: Configurando o Admin

O Django vem com um sistema de administração embutido que facilita a gestão dos modelos de dados. Esse sistema é altamente personalizável e muito útil para visualizar, adicionar, editar e excluir dados diretamente da interface administrativa.

Configurando o Django Admin

Para registrar o modelo `Item` no admin, você deve editar o arquivo `admin.py` da sua aplicação. A seguir, mostramos como fazer isso.

```
python
Copiar código
# admin.py em app_crud
from django.contrib import admin
from .models import Item

# Registrando o modelo Item no admin
admin.site.register(Item)
```

Após adicionar este código, você poderá acessar o painel administrativo do Django, onde o modelo `Item` será exibido, permitindo que você crie, edite e exclua itens diretamente por essa interface.

Para acessar o admin, você precisa criar um superusuário:

```
bash
Copiar código
# Criando um superusuário
python manage.py createsuperuser
```

Depois de criar o superusuário, você pode acessar o painel administrativo em `http://127.0.0.1:8000/admin` e fazer login com as credenciais do superusuário.

Capítulo 4: Criando as Vistas e URLs

Definindo URLs

As URLs no Django são definidas em arquivos chamados `urls.py`. Vamos começar criando as URLs para as operações de CRUD em nosso aplicativo `app_crud`.

Primeiro, crie o arquivo `urls.py` dentro do diretório `app_crud` se ele ainda não existir.

```
python
Copiar código
# urls.py em app_crud
from django.urls import path
from . import views

urlpatterns = [
    path('', views.ItemListView.as_view(), name='item-list'),
    path('item/<int:pk>/', views.ItemDetailView.as_view(), name='item-detail'),
    path('item/new/', views.ItemCreateView.as_view(), name='item-create'),
    path('item/<int:pk>/edit/', views.ItemUpdateView.as_view(), name='item-
edit'),
```

```
    path('item/<int:pk>/delete/', views.ItemDeleteView.as_view(), name='item-
delete'),
]
```

Agora, inclua essas URLs no arquivo `urls.py` do projeto principal para que elas sejam reconhecidas.

```
python
Copiar código
# urls.py em meu_projeto
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('app_crud.urls')),
]
```

Criando Vistas

Vamos criar vistas baseadas em classes para cada operação de CRUD. As vistas baseadas em classes fornecem uma maneira organizada e reutilizável de lidar com as operações comuns.

Listar Itens

```
python
Copiar código
# views.py em app_crud
from django.views.generic import ListView, DetailView, CreateView, UpdateView,
DeleteView
from .models import Item

class ItemListView(ListView):
    model = Item
    template_name = 'item_list.html'
```

A `ItemListView` usará o template `item_list.html` para exibir a lista de itens. O Django fornece automaticamente o contexto necessário para renderizar a lista de objetos do modelo `Item`.

Detalhar Item

```
python
Copiar código
class ItemDetailView(DetailView):
    model = Item
    template_name = 'item_detail.html'
```

A `ItemDetailView` exibirá os detalhes de um item específico usando o template `item_detail.html`.

Criar Item

```
python
Copiar código
class ItemCreateView(CreateView):
    model = Item
    fields = ['nome', 'descricao', 'preco']
    template_name = 'item_form.html'
```

A `ItemCreateView` fornecerá um formulário para criar um novo item usando o template `item_form.html`. Os campos do formulário são especificados na lista `fields`.

Atualizar Item

```
python
Copiar código
class ItemUpdateView(UpdateView):
    model = Item
    fields = ['nome', 'descricao', 'preco']
    template_name = 'item_form.html'
```

A `ItemUpdateView` usará o mesmo template `item_form.html` para editar um item existente.

Excluir Item

```
python
Copiar código
class ItemDeleteView(DeleteView):
    model = Item
    success_url = '/'
    template_name = 'item_confirm_delete.html'
```

A `ItemDeleteView` exibirá um formulário de confirmação antes de excluir um item e redirecionará para a URL de sucesso (`success_url`) após a exclusão.

Capítulo 5: Desenvolvendo os Templates

Os templates no Django são usados para renderizar HTML com dados dinâmicos. Vamos criar os templates necessários para as operações de CRUD.

Template Base

O template base serve como um layout comum para todas as outras páginas, contendo elementos como cabeçalho, rodapé e navegação.

```
html
Copiar código
<!-- Template base (base.html) -->
<!DOCTYPE html>
<html>
<head>
    <title>CRUD com Django</title>
</head>
<body>
    <header>
        <h1>CRUD com Django</h1>
        <nav>
            <ul>
                <li><a href="{% url 'item-list' %}">Home</a></li>
                <li><a href="{% url 'item-create' %}">Criar Novo Item</a></li>
            </ul>
        </nav>
    </header>
    <main>
        {% block content %}
        {% endblock %}
    </main>
    <footer>
```

```
        <p>&copy; 2023 CRUD com Django</p>
    </footer>
</body>
</html>
```

Listagem de Itens

```
html
Copiar código
<!-- Listagem de Itens (item_list.html) -->
{% extends 'base.html' %}

{% block content %}
    <h2>Lista de Itens</h2>
    <ul>
        {% for item in object_list %}
            <li>
                <a href="{% url 'item-detail' item.pk %}">{{ item.nome }}</a>
                - <a href="{% url 'item-edit' item.pk %}">Editar</a>
                - <a href="{% url 'item-delete' item.pk %}">Excluir</a>
            </li>
        {% endfor %}
    </ul>
{% endblock %}
```

Detalhes do Item

```
html
Copiar código
<!-- Detalhes do Item (item_detail.html) -->
{% extends 'base.html' %}

{% block content %}
    <h2>{{ object.nome }}</h2>
    <p>{{ object.descricao }}</p>
    <p>Preço: {{ object.preco }}</p>
    <p>Criado em: {{ object.criado_em }}</p>
    <a href="{% url 'item-edit' object.pk %}">Editar</a>
    <a href="{% url 'item-delete' object.pk %}">Excluir</a>
{% endblock %}
```

Formulário de Criação e Edição

```
html
Copiar código
<!-- Formulário de Criação e Edição (item_form.html) -->
{% extends 'base.html' %}

{% block content %}
    <h2>{% if object %}Editar Item{% else %}Criar Novo Item{% endif %}</h2>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">{% if object %}Atualizar{% else %}Criar{% endif
%}</button>
    </form>
{% endblock %}
```

Confirmação de Exclusão

```
html
Copiar código
<!-- Confirmação de Exclusão (item_confirm_delete.html) -->
{% extends 'base.html' %}

{% block content %}
    <h2>Confirmar Exclusão</h2>
    <p>Tem certeza que deseja excluir "{{ object.nome }}"?</p>
    <form method="post">
        {% csrf_token %}
        <button type="submit">Sim, excluir</button>
        <a href="{% url 'item-list' %}">Cancelar</a>
    </form>
{% endblock %}
```

Capítulo 6: Implementando Funcionalidades de CRUD

Operação de Criação

Para criar um novo item, usamos a vista `ItemCreateView` que renderiza um formulário para entrada de dados.

Vistas Baseadas em Classes para Criação

```
python
Copiar código
class ItemCreateView(CreateView):
    model = Item
    fields = ['nome', 'descricao', 'preco']
    template_name = 'item_form.html'
```

O formulário é renderizado usando o template `item_form.html`. Quando o formulário é enviado, o Django valida e salva os dados no banco de dados.

Operação de Leitura

A operação de leitura inclui tanto a listagem de todos os itens quanto a exibição de detalhes de um item específico.

Listar Itens

```
python
Copiar código
class ItemListView(ListView):
    model = Item
    template_name = 'item_list.html'
```

Exibir Detalhes do Item

```
python
Copiar código
class ItemDetailView(DetailView):
    model = Item
    template_name = 'item_detail.html'
```

Operação de Atualização

Para editar um item, usamos a vista `ItemUpdateView`.

```
python
Copiar código
class ItemUpdateView(UpdateView):
    model = Item
    fields = ['nome', 'descricao', 'preco']
    template_name = 'item_form.html'
```

A mesma lógica de formulário de criação é usada aqui, mas com os dados existentes do item carregados no formulário.

Operação de Exclusão

Para excluir um item, usamos a vista `ItemDeleteView`.

```
python
Copiar código
class ItemDeleteView(DeleteView):
    model = Item
    success_url = '/'
    template_name = 'item_confirm_delete.html'
```

A `ItemDeleteView` exibe uma página de confirmação antes de excluir o item e redireciona para a página inicial após a exclusão.

Capítulo 7: Melhorando a Aplicação

Validação de Formulários

Podemos adicionar validação personalizada ao nosso formulário de criação/edição para garantir que os dados inseridos são válidos.

Validação Personalizada

```
python
Copiar código
# forms.py em app_crud
from django import forms
from .models import Item

class ItemForm(forms.ModelForm):
    class Meta:
        model = Item
        fields = ['nome', 'descricao', 'preco']

    def clean_preco(self):
        preco = self.cleaned_data.get('preco')
        if preco <= 0:
            raise forms.ValidationError('O preço deve ser maior que zero.')
        return preco
```

Autenticação e Autorização

Para garantir que apenas usuários autenticados possam criar, editar ou excluir itens, usamos mixins de autenticação.

Restringindo Acesso às Vistas

```
python
Copiar código
# views.py em app_crud
from django.contrib.auth.mixins import LoginRequiredMixin

class ItemCreateView(LoginRequiredMixin, CreateView):
    model = Item
    form_class = ItemForm
    template_name = 'item_form.html'
    login_url = '/login/'

class ItemUpdateView(LoginRequiredMixin, UpdateView):
    model = Item
    form_class = ItemForm
    template_name = 'item_form.html'
    login_url = '/login/'

class ItemDeleteView(LoginRequiredMixin, DeleteView):
    model = Item
    success_url = '/'
    template_name = 'item_confirm_delete.html'
    login_url = '/login/'
```

Capítulo 8: Testando a Aplicação

Testes Unitários

Os testes unitários garantem que cada parte do seu código funcione conforme esperado.

Testando o Modelo

```
python
Copiar código
# tests.py em app_crud
from django.test import TestCase
from .models import Item

class ItemModelTest(TestCase):

    def setUp(self):
        Item.objects.create(nome='Item Teste', descricao='Descrição do item
teste', preco=10.0)

    def test_item_criado(self):
        item = Item.objects.get(nome='Item Teste')
        self.assertEqual(item.descricao, 'Descrição do item teste')
```

Testes de Integração

Os testes de integração verificam se diferentes partes da aplicação funcionam juntas corretamente.

Testando o Fluxo Completo do CRUD

```
python
Copiar código
# tests.py em app_crud
from django.urls import reverse

class ItemCRUDTest(TestCase):

    def setUp(self):
        self.item = Item.objects.create(nome='Item Teste', descricao='Descrição
do item teste', preco=10.0)

    def test_create_item(self):
        response = self.client.post(reverse('item-create'), {'nome': 'Novo
Item', 'descricao': 'Descrição', 'preco': 20.0})
        self.assertEqual(response.status_code, 302)

    def test_read_item(self):
        response = self.client.get(reverse('item-detail', args=[self.item.pk]))
        self.assertContains(response, self.item.nome)

    def test_update_item(self):
        response = self.client.post(reverse('item-edit', args=[self.item.pk]),
{'nome': 'Item Atualizado', 'descricao': 'Descrição Atualizada', 'preco': 15.0})
        self.assertEqual(response.status_code, 302)
        self.item.refresh_from_db()
        self.assertEqual(self.item.nome, 'Item Atualizado')

    def test_delete_item(self):
        response = self.client.post(reverse('item-delete', args=[self.item.pk]))
        self.assertEqual(response.status_code, 302)
        self.assertFalse(Item.objects.filter(pk=self.item.pk).exists())
```

Conclusão

Resumo do Desenvolvimento

Neste eBook, abordamos o desenvolvimento de um CRUD completo com Django, desde a configuração do ambiente até a implementação de funcionalidades avançadas e testes. Você aprendeu a:

- Configurar seu ambiente de desenvolvimento.
- Criar e configurar um projeto e uma aplicação Django.
- Definir modelos, vistas, URLs e templates.
- Implementar operações de CRUD.
- Melhorar a aplicação com validação de formulários e autenticação.
- Testar a aplicação usando testes unitários e de integração.

Próximos Passos

Recursos adicionais para aprendizado:

- Documentação oficial do Django
- Tutoriais e cursos online
- Comunidades e fóruns de Django

Sugestões de projetos para prática:

- Expandir o CRUD para incluir funcionalidades adicionais, como pesquisa e paginação.
- Integrar a aplicação com uma API externa.
- Implementar testes mais abrangentes e automação de testes contínua.