Tutorial: Criando um Projeto Django para Explicar Interação com Banco de Dados

- **Instalação e configuração inicial**: Incluindo setup do ambiente virtual, instalação do Django e configuração do banco de dados (usaremos SQLite por simplicidade, mas explicarei como trocar para PostgreSQL ou MySQL).
- **Modelos (Models)**: Definição de tabelas no banco de dados.
- **Views e Forms**: Lógica para consultas (read), cadastros (create), atualizações (update) e exclusões (delete) – o famoso CRUD.
- **Templates**: Telas HTML com CSS para estilização e JavaScript para interações dinâmicas (ex.: validação de formulários, busca em tempo real).
- **Administração**: Uso do admin do Django para demonstrar interações básicas.

Essa aplicação será mínima, mas completa, para ilustrar conceitos. Ao final, você terá um projeto rodando localmente.

Pré-requisitos

- Python 3.8+ instalado (verifique com python --version).
- Conhecimento básico de terminal/comando de linha.
- Editor de código (recomendo VS Code ou PyCharm).

Passo 1: Instalação e Configuração Inicial

1.1 Criar Ambiente Virtual

Para isolar as dependências do projeto:

- Abra o terminal e crie uma pasta para o projeto: mkdir django-banco-exemplo && cd django-banco-exemplo.
- Crie o ambiente virtual: python -m venv venv.
- Ative-o:
    - Windows: venv\Scripts\activate.
    - Linux/Mac: source venv/bin/activate.

1.2 Instalar Django

- No terminal (com venv ativado): pip install django.
- Verifique: django-admin --version (deve mostrar algo como 5.1 ou superior, dependendo da data).

1.3 Criar o Projeto Django

- django-admin startproject biblioteca . (o ponto . cria na pasta atual).
- Isso gera arquivos como manage.py, biblioteca/settings.py, etc.

1.4 Configurar o Banco de Dados

No arquivo biblioteca/settings.py, procure pela seção DATABASES. Por padrão, usa SQLite (fácil para testes):

python

```
DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': BASE_DIR / 'db.sqlite3',

    }

}
```

- **Explicação para alunos**: O Django usa ORM (Object-Relational Mapping) para interagir com o BD sem SQL puro. Aqui, configuramos o engine (SQLite). Para PostgreSQL, instale pip install psycopg2 e mude para:
  python

```
'ENGINE': 'django.db.backends.postgresql',
```

-

```
'NAME': 'seu_banco',
```

-

```
'USER': 'seu_usuario',
```

-

```
'PASSWORD': 'sua_senha',
```

-

```
'HOST': 'localhost',
```

-
- `'PORT': '5432',`
- Rode migrações iniciais: python manage.py migrate (cria tabelas padrão do Django, como usuários).

1.5 Criar uma App Dentro do Projeto

- python manage.py startapp livros.
- Adicione 'livros' à lista INSTALLED_APPS em settings.py.

Passo 2: Definindo Modelos (Interação com Banco de Dados)

Em livros/models.py, crie o modelo para "Livro":

```
from django.db import models

class Livro(models.Model):

   titulo = models.CharField(max_length=200)

   autor = models.CharField(max_length=100)

   ano_publicacao = models.IntegerField()

   disponivel = models.BooleanField(default=True)

   def __str__(self):

       return self.titulo
```

- **Explicação**: Cada classe herda de models.Model e representa uma tabela. Campos como CharField mapeiam para colunas SQL. O __str__ é para representação amigável.
- Crie migrações: python manage.py makemigrations.
- Aplique: python manage.py migrate (isso cria a tabela livros_livro no BD).

Use o shell do Django (python manage.py shell) para interagir:

```
from livros.models import Livro

Livro.objects.create(titulo='1984', autor='George Orwell',
ano_publicacao=1949)

livros = Livro.objects.all()  # Consulta todos

print(livros)
```

Passo 3: Configurando o Admin (Para Demonstração Rápida de CRUD)

Em livros/admin.py:

```
from django.contrib import admin

from .models import Livro



admin.site.register(Livro)
```

- Crie um superusuário: python manage.py createsuperuser.
- Rode o server: python manage.py runserver.
- Acesse http://127.0.0.1:8000/admin/ e logue. Aqui, você pode cadastrar, consultar, editar e deletar livros via interface admin.
- **Explicação**: O admin é uma ferramenta pronta do Django para CRUD básico, ideal para mostrar interações sem código extra.

Passo 4: Criando Views, Forms e URLs (CRUD Personalizado)

4.1 Forms

Em livros/forms.py (crie o arquivo):

```
from django import forms

from .models import Livro



class LivroForm(forms.ModelForm):

    class Meta:

        model = Livro

        fields = ['titulo', 'autor', 'ano_publicacao', 'disponivel']
```

- **Explicação**: Forms mapeiam modelos para HTML, validam dados e salvam no BD.

4.2 Views

Em livros/views.py:

```
from django.shortcuts import render, redirect, get_object_or_404
```

```python
from .models import Livro
from .forms import LivroForm

def lista_livros(request):
    livros = Livro.objects.all()  # Consulta ao BD
    return render(request, 'livros/lista.html', {'livros': livros})

def novo_livro(request):
    if request.method == 'POST':
        form = LivroForm(request.POST)
        if form.is_valid():
            form.save()  # Salva no BD
            return redirect('lista_livros')
    else:
        form = LivroForm()
    return render(request, 'livros/form.html', {'form': form})

def editar_livro(request, id):
    livro = get_object_or_404(Livro, pk=id)
    if request.method == 'POST':
```

```
        form = LivroForm(request.POST, instance=livro)
        if form.is_valid():
            form.save()
            return redirect('lista_livros')
    else:
        form = LivroForm(instance=livro)
    return render(request, 'livros/form.html', {'form': form})


def deletar_livro(request, id):
    livro = get_object_or_404(Livro, pk=id)
    if request.method == 'POST':
        livro.delete()  # Deleta do BD
        return redirect('lista_livros')
    return render(request, 'livros/deletar.html', {'livro': livro})
```

- **Explicação**:
  - lista_livros: Consulta (Read) com objects.all().
  - novo_livro: Cadastro (Create) com form.save().
  - editar_livro: Atualização (Update).
  - deletar_livro: Exclusão (Delete).

4.3 URLs

Em livros/urls.py (crie o arquivo):

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

```
urlpatterns = [

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

    path('novo/', views.novo_livro, name='novo_livro'),

    path('editar/<int:id>/', views.editar_livro, name='editar_livro'),

    path('deletar/<int:id>/', views.deletar_livro, name='deletar_livro'),

]
```

Em biblioteca/urls.py, adicione:

```
from django.contrib import admin

from django.urls import path, include



urlpatterns = [

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

    path('livros/', include('livros.urls')),

]
```

Passo 5: Templates (Telas com CSS e JavaScript)

Crie a pasta livros/templates/livros/ e os arquivos abaixo.

5.1 Base Template (base.html) para Herança

```
<!DOCTYPE html>

<html lang="pt-br">
```

```
<head>

    <meta charset="UTF-8">

    <title>{% block title %}Biblioteca{% endblock %}</title>

    <style>

        body { font-family: Arial, sans-serif; background-color: #f4f4f4;
margin: 20px; }

        h1 { color: #333; }

        table { width: 100%; border-collapse: collapse; margin-bottom: 20px;
}

        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }

        th { background-color: #4CAF50; color: white; }

        form { background: white; padding: 20px; border-radius: 5px;
box-shadow: 0 0 10px rgba(0,0,0,0.1); }

        input, button { margin: 10px 0; padding: 8px; width: 100%;
box-sizing: border-box; }

        button { background-color: #4CAF50; color: white; border: none;
cursor: pointer; }

        button:hover { background-color: #45a049; }

    </style>

</head>

<body>

    <h1>Gerenciamento de Livros</h1>
```

```
   <a href="{% url 'lista_livros' %}">Lista de Livros</a> | <a href="{% url
'novo_livro' %}">Adicionar Novo</a>

   <hr>

   {% block content %}{% endblock %}

</body>

</html>
```

- **CSS Explicação**: Estilos básicos para tabelas, forms e botões. Demonstre como CSS melhora a UI.

5.2 Lista de Livros (lista.html)

```
{% extends 'livros/base.html' %}



{% block content %}

   <input type="text" id="busca" placeholder="Busque por título..."
onkeyup="buscarLivros()">

   <table id="tabelaLivros">


<tr><th>Título</th><th>Autor</th><th>Ano</th><th>Disponível</th><th>Ações</
th></tr>

       {% for livro in livros %}

           <tr>

               <td>{{ livro.titulo }}</td>

               <td>{{ livro.autor }}</td>

               <td>{{ livro.ano_publicacao }}</td>
```

```
                <td>{% if livro.disponivel %}Sim{% else %}Não{% endif
%}</td>

                <td>

                    <a href="{% url 'editar_livro' livro.id %}">Editar</a> |

                    <a href="{% url 'deletar_livro' livro.id %}">Deletar</a>

                </td>

            </tr>

        {% endfor %}

    </table>

    <script>

        function buscarLivros() {

            let input =
document.getElementById('busca').value.toLowerCase();

            let tabela = document.getElementById('tabelaLivros');

            let linhas = tabela.getElementsByTagName('tr');

            for (let i = 1; i < linhas.length; i++) {

                let titulo =
linhas[i].getElementsByTagName('td')[0].textContent.toLowerCase();

                linhas[i].style.display = titulo.includes(input) ? '' :
'none';

            }

        }
```

```
    </script>

{% endblock %}
```

- **JavaScript Explicação**: Função buscarLivros() filtra a tabela em tempo real. Mostre como JS interage com DOM sem recarregar a página.

5.3 Formulário (form.html)

```
{% extends 'livros/base.html' %}

{% block content %}

    <form method="post">

        {% csrf_token %}

        {{ form.as_p }}

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

    </form>

    <script>

        // Exemplo de validação JS

        document.querySelector('form').addEventListener('submit',
function(e) {

            let ano =
document.querySelector('input[name="ano_publicacao"]').value;

            if (ano < 1900 || ano > new Date().getFullYear()) {

                alert('Ano de publicação inválido!');

                e.preventDefault();

            }
```

```
        });
    </script>
{% endblock %}
```

- **Explicação**: {{ form.as_p }} renderiza o form. JS valida o ano antes de enviar.

5.4 Deletar (deletar.html)

```
{% extends 'livros/base.html' %}



{% block content %}
    <p>Confirma deletar "{{ livro.titulo }}"?</p>
    <form method="post">
        {% csrf_token %}
        <button type="submit">Sim</button>
        <a href="{% url 'lista_livros' %}">Cancelar</a>
    </form>
{% endblock %}
```

Passo 6: Rodando e Testando

- python manage.py runserver.
- Acesse http://127.0.0.1:8000/livros/.
- Teste: Adicione livros, busque, edite, delete. Use o admin para ver o BD diretamente.