Apostila 3: Models e o Django Admin

O "M" do MVT: O Cérebro do seu Site

1. Introdução

O que é um "Model"?

Um **Model** (Modelo) é a sua "planta baixa" para o banco de dados. É uma classe Python que define *exatamente* quais informações você quer salvar e como elas devem ser.

- Você quer salvar **Usuários**? Você cria um Model de Usuário.
- Você quer salvar **Produtos**? Você cria um Model de Produto.
- Você quer salvar **Posts de um blog**? Você cria um Model de Post.

O que é um ORM (Object-Relational Mapper)?

Este é o "superpoder" do Django. Em vez de escrever comandos SQL complicados (como CREATE TABLE ..., INSERT INTO ...), você simplesmente escreve uma **classe Python**.

O **ORM** do Django "traduz" essa classe Python e gera todo o SQL necessário para você.

- **Você escreve (Python):** class Produto(...):
- **O Django entende (SQL):** CREATE TABLE produto (...);

Nosso Objetivo: Vamos criar um Model para uma lista de **Tarefas** (To-Do List). Cada tarefa terá um título, um status (concluída ou não) e uma data de criação. Em seguida, vamos usar o painel de administração secreto do Django para adicionar, editar e excluir essas tarefas.

---

2. Módulo 1: Definindo seu Primeiro Model (20 min)

Vamos definir a estrutura das nossas tarefas.

Passo 1: Edite o core/models.py
Até agora, este arquivo estava vazio. Abra core/models.py e adicione o seguinte código:

Python

```python
# core/models.py

from django.db import models

# Cada classe aqui é "traduzida" para uma tabela no banco de dados
class Tarefa(models.Model):
    # Um campo de texto curto, com máximo de 200 caracteres
    titulo = models.CharField(max_length=200)

    # Um campo booleano (verdadeiro/falso), que por padrão é Falso
    concluida = models.BooleanField(default=False)

    # Um campo de data e hora.
    # auto_now_add=True: Salva a data e hora EXATAMENTE quando o objeto é criado
    criada_em = models.DateTimeField(auto_now_add=True)

    # Este é um "método mágico" do Python.
    # Ele diz ao Django como "chamar" um objeto Tarefa.
    # Em vez de "Tarefa object (1)", ele mostrará o título da tarefa.
    # Isso é EXTREMAMENTE útil no painel de administração.
    def __str__(self):
        return self.titulo
```

Parabéns, você acabou de criar sua "planta baixa".

---

3. Módulo 2: Migrations (O Plano de Obra) (20 min)

Você desenhou a planta (models.py), mas o banco de dados (db.sqlite3) ainda não sabe dela. Precisamos "enviar" essa planta para ele.

Esse processo é feito em dois comandos:

1. **makemigrations**: O Django olha seus models.py, compara com o que *já existe* no banco

de dados, e **cria um arquivo de "plano de obra"** (uma "migração") com as instruções do que precisa mudar (ex: "criar esta nova tabela").
2. **migrate**: O Django pega todos os "planos de obra" que ainda não foram executados e **aplica eles** no banco de dados.

Passo 1: makemigrations (O Plano)
Pare seu servidor (Ctrl+C) e rode no terminal:

Bash

```
# É uma boa prática especificar qual app você quer "migrar"
python manage.py makemigrations core
```

Você verá uma saída parecida com esta:

```
Migrations for 'core':
  core\migrations\0001_initial.py
    - Create model Tarefa
```

Isso criou o arquivo 0001_initial.py dentro de core/migrations/. Ele é o "plano de obra" escrito em Python.

Passo 2: migrate (A Construção)
Agora, vamos mandar o Django "construir":

Bash

```
python manage.py migrate
```

A saída será longa! Ele vai aplicar não só a *sua* migração, mas também todas as migrações que o Django já usa por padrão (para usuários, sessões, admin, etc.).

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, core, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  ...
  Applying core.0001_initial... OK  <-- AQUI ESTÁ A NOSSA!
  ...
```

Resultado: Se você olhar na pasta raiz do seu projeto, verá um novo arquivo: **db.sqlite3**. Este é o seu banco de dados! A tabela core_tarefa agora existe dentro dele.

---

4. Módulo 3: O Django Admin (O Superpoder) (30 min)

Esta é a "bateria inclusa" mais famosa do Django. Ele te dá um site de administração completo e seguro para gerenciar seus dados, com zero esforço.

Passo 1: Crie um Superusuário
Primeiro, precisamos de um login para acessar o painel.

Bash

```
python manage.py createsuperuser
```

Siga as instruções. Crie um nome de usuário (ex: admin), um e-mail (pode ser falso, como admin@admin.com) e uma senha.

Passo 2: Registre seu Model no Admin
O Django não expõe seus Models no Admin por padrão (por segurança). Você precisa registrar manualmente quais Models você quer gerenciar.
Abra o arquivo **core/admin.py** e adicione:

Python

```
# core/admin.py

from django.contrib import admin
from .models import Tarefa  # 1. Importe seu Model

# 2. "Registre" seu Model no site de administração
admin.site.register(Tarefa)
```

É só isso. Duas linhas de código.

Passo 3: Explore o Admin!
Vamos ver a mágica. Rode o servidor:
python manage.py runserver
Agora, acesse a URL: **http://127.0.0.1:8000/admin**

Use o login e senha que você criou. Você verá o painel de administração do Django. Em "CORE", você verá "Tarefas".

- **Clique em "Tarefas".**
- **Clique em "ADD TAREFA +"** no canto superior direito.
- Note que o formulário foi **criado automaticamente** para você!
- Crie algumas tarefas de exemplo:
  - "Lavar a louça" (deixe concluida desmarcado)
  - "Estudar Django" (deixe concluida desmarcado)
  - "Comprar pão" (marque concluida como True)
- Clique em "Save".

Veja como a lista aparece com os nomes "Lavar a louça", etc. Isso é por causa do def __str__(self): que escrevemos no Model. Se não tivéssemos feito isso, você veria Tarefa object (1), Tarefa object (2), etc.

---

5. Módulo 4: Fechando o Ciclo (Dados do BD no Template) (20 min)

Temos dados no banco de dados. Temos um template. Vamos ligar os dois!

Passo 1: Busque os dados na view
Vamos editar core/views.py para buscar todas as tarefas do banco de dados e enviá-las para o template.

```
Python
```

```
# core/views.py

from django.shortcuts import render
from .models import Tarefa  # 1. Importe o Model Tarefa

def home(request):

    # 2. Use o ORM para buscar os dados!
    # Tarefa.objects.all() significa: "Pegue todas as linhas da tabela Tarefa"
    todas_as_tarefas = Tarefa.objects.all()

    # 3. Atualize o contexto
    context = {
        'nome_usuario': 'Júnior',
        'tecnologias': ['Python', 'Django', 'Models', 'Admin'],
        'tarefas': todas_as_tarefas  # 4. Adicione as tarefas ao contexto
    }

    return render(request, 'home.html', context)
```

Passo 2: Exiba os dados no Template
Agora, vamos editar templates/home.html para mostrar essa lista de tarefas.
Abra templates/home.html e, dentro do {% block content %}, adicione o seguinte (pode ser abaixo da lista de tecnologias):

```
    HTML
```

```
...
{% block content %}
    <h1>Olá, {{ nome_usuario }}!</h1>

    ... (seu código antigo) ...

    <hr> <h2>Minha Lista de Tarefas (do Banco de Dados):</h2>

    <ul>
        {% for tarefa in tarefas %}
            <li>
```

```
            {{ tarefa.titulo }}
            {% if tarefa.concluida %}
                (✓ Concluída)
            {% else %}
                (☐ Pendente)
            {% endif %}
            <br>
            <small>Criada em: {{ tarefa.criada_em }}</small>
        </li>
    {% empty %}
        <li>Você ainda não tem tarefas cadastradas.</li>
    {% endfor %}
</ul>

{% endblock %}
```

- {% for tarefa in tarefas %}: Itera sobre a lista que veio da view.
- {{ tarefa.titulo }}: Acessa o atributo titulo do objeto tarefa.
- {% empty %}: Um bônus útil da tag {% for %}.

Passo 3: Teste!
Recarregue sua página principal: http://127.0.0.1:8000.
Você verá a lista de tarefas que você cadastrou no **Django Admin**!

Experimente:

1. Vá até o Admin (/admin).
2. Adicione uma nova tarefa (ex: "Dominar o mundo").
3. Volte para a Home (/) e recarregue a página.
4. A nova tarefa aparecerá **instantaneamente**.