Apostila 8: Múltiplos Apps e Integração

A Arquitetura Correta de um Projeto Django

1. Introdução

Nosso Cenário:
Nossas Tarefas atualmente "flutuam" no ar. Elas pertencem a um User, mas só isso. Vamos organizá-las em Projetos.
A Separação de Responsabilidades:

1. **Novo App (projects):** Sua *única* responsabilidade será gerenciar "Projetos". Ele terá um Model de Project.
2. **App Antigo (core):** Sua responsabilidade é gerenciar Tarefas.

A Integração (A "Conversa"):
Vamos criar uma relação onde cada Tarefa (do app core) deve pertencer a um Project (do app projects).
Nosso Objetivo: Criar um segundo app, relacionar os Models dos dois apps, e modificar nosso formulário de tarefas para incluir um seletor de projetos que mostre *apenas* os projetos do usuário logado.

---

2. Módulo 1: O Novo App (projects)

Passo 1: Crie o app
Pare o servidor. No terminal, na raiz do projeto, rode o comando startapp:

Bash

python manage.py startapp projects

Sua estrutura de pastas agora tem dois apps:

```
core/
meuprojeto/
projects/      <-- NOSSO NOVO APP
templates/
static/
...
```

Passo 2: Registre o app
Como fizemos na Apostila 1, precisamos "avisar" o settings.py sobre o novo app.
Abra meuprojeto/settings.py e adicione projects à lista INSTALLED_APPS:

Python

```
# meuprojeto/settings.py
INSTALLED_APPS = [
    ...
    'core',
    'projects',  # <-- ADICIONE ESTA LINHA
]
```

---

3. Módulo 2: Modelagem (A Relação entre Apps)

Agora, vamos criar o Model de Project e, em seguida, conectar nosso Model de Tarefa a ele.

Passo 1: Crie o Model de Project
Abra o novo arquivo projects/models.py e defina o Model:

Python

```
# projects/models.py

from django.db import models
from django.contrib.auth.models import User

# Um Projeto também pertence a um usuário
class Project(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    titulo = models.CharField(max_length=100)

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

Passo 2: Conecte a Tarefa ao Project
Abra o arquivo antigo core/models.py. Vamos importar o Model que acabamos de criar e adicioná-lo à Tarefa.

Python

```
# core/models.py

from django.db import models
from django.contrib.auth.models import User
# 1. Importe o Model do outro app!
from projects.models import Project

class Tarefa(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)

    # 2. ADICIONE A NOVA RELAÇÃO
    # Cada Tarefa agora DEVE estar ligada a um Projeto.
    # 'related_name' é uma boa prática para consultas inversas.
    project = models.ForeignKey(Project, on_delete=models.CASCADE, related_name='tarefas')

    titulo = models.CharField(max_length=200)
    concluida = models.BooleanField(default=False)
    criada_em = models.DateTimeField(auto_now_add=True)
```

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

---

4. Módulo 3: Migrations (Recomeçando o Banco)

Assim como na Apostila 6, fizemos uma mudança estrutural (adicionamos um ForeignKey obrigatório, project).

A forma mais limpa de lidar com isso em nosso tutorial é **recomeçar o banco de dados**.

Passo 1: Limpe tudo

1. **Pare o servidor** (Ctrl+C).
2. **Delete o arquivo db.sqlite3** na raiz do projeto.
3. **Delete** a pasta core/migrations/ (ou tudo dentro dela, exceto __init__.py).
4. **Delete** a pasta projects/migrations/ (ou tudo dentro dela, exceto __init__.py).

Passo 2: Crie as novas migrações (para os dois apps)
Agora, o Django vai ler os dois apps e criar os "planos de obra" em conjunto.

```
Bash
```

```
# Você pode especificar os apps que quer migrar
python manage.py makemigrations core projects
```

Você verá que ele criou 0001_initial.py em *ambos* os apps.

Passo 3: Execute e Crie seu Usuário

```
Bash
```

```
# Crie o novo 'db.sqlite3' com a estrutura correta
python manage.py migrate

# Crie seu usuário para poder acessar o site
```

```
python manage.py createsuperuser
```

---

5. Módulo 4: O Admin (Gerenciando Projetos)

Vamos rapidamente registrar o Project no Admin para podermos criar alguns projetos de teste.

Passo 1: Crie o arquivo projects/admin.py
O startapp não cria este arquivo por padrão. Crie-o:

```
projects/
  admin.py    <-- CRIE ESTE ARQUIVO
  ...
```

Passo 2: Registre o Project
Abra projects/admin.py e adicione o código (vamos usar as técnicas da Apostila 7):

Python

```
# projects/admin.py

from django.contrib import admin
from .models import Project

# Reforçando a Apostila 7
class ProjectAdmin(admin.ModelAdmin):
  list_display = ('titulo', 'user')
  search_fields = ('titulo', 'user__username')
  list_filter = ('user',)

admin.site.register(Project, ProjectAdmin)
```

Não se esqueça de registrar Tarefa no core/admin.py também (pode usar o código da Apostila 7).

---

6. Módulo 5: A Integração (Formulário e View)

Esta é a parte crucial. Nosso TarefaForm precisa ser modificado para mostrar um campo de seleção de Project.

Mas ele não pode mostrar *todos* os projetos do site; ele deve mostrar apenas os projetos do **usuário logado**.

Passo 1: Modifique o TarefaForm (em core/forms.py)
Precisamos de uma lógica avançada: vamos "passar" o user da view para o form e usá-lo para filtrar o campo project.

Python

```
# core/forms.py

from django import forms
from .models import Tarefa
from projects.models import Project # Importe o Project

class TarefaForm(forms.ModelForm):

    # 1. Vamos sobrescrever o construtor do formulário
    def __init__(self, *args, **kwargs):
        # 2. Capture o 'user' que será passado pela view
        user = kwargs.pop('user', None)

        # 3. Chame o construtor original (pai)
        super(TarefaForm, self).__init__(*args, **kwargs)

        # 4. A MÁGICA:
        # Se o 'user' foi passado...
        if user:
            # ...filtre o 'queryset' (a lista de opções) do campo 'project'
            # para mostrar apenas os projetos onde o 'user' é o usuário logado.
            self.fields['project'].queryset = Project.objects.filter(user=user)

    class Meta:
```

```
        model = Tarefa
        # 5. Adicione 'project' aos campos do formulário
        fields = ['titulo', 'project']
```

Passo 2: Modifique a View home (em core/views.py)
Agora, precisamos passar o request.user para o TarefaForm que acabamos de modificar.

Python

```
# core/views.py
...

@login_required
def home(request):
    if request.method == 'POST':
        # 1. Passe o 'user' logado para o formulário no POST
        form = TarefaForm(request.POST, user=request.user)

        if form.is_valid():
            tarefa = form.save(commit=False)
            tarefa.user = request.user
            tarefa.save()
            # O 'project' já é salvo automaticamente pelo form.save()
            return redirect('home')
    else:
        # 2. Passe o 'user' logado para o formulário no GET
        form = TarefaForm(user=request.user)

    todas_as_tarefas = Tarefa.objects.filter(user=request.user).order_by('-criada_em')

    context = {
        'nome_usuario': request.user.username,
        'tarefas': todas_as_tarefas,
        'form': form,
    }

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

---

7. Módulo 6: O Template (Exibindo a Relação)

Nosso formulário {{ form.as_p }} já vai renderizar o novo campo <select> de "Project" automaticamente.

Vamos apenas atualizar nossa *lista* de tarefas para mostrar a qual projeto a tarefa pertence.

Passo 1: Edite templates/home.html
Modifique o loop {% for tarefa in tarefas %}:

HTML

```
...
    <ul style="list-style: none; padding: 0;">
        {% for tarefa in tarefas %}
            <li style="margin-bottom: 15px; border: 1px solid #ccc; padding: 10px;">

                <small style="font-weight: bold; color: #555;">
                    Projeto: {{ tarefa.project.titulo }}
                </small>
                <br>

                {{ tarefa.titulo }}

                {% if tarefa.concluida %}
                    <strong style="color: green;">(✓ Concluída)</strong>
                {% else %}
                    <strong style="color: orange;">(☐ Pendente)</strong>
                {% endif %}
                ... (resto do código dos botões) ...
            </li>
        {% empty %}
            <li>Você ainda não tem tarefas cadastradas.</li>
        {% endfor %}
    </ul>
...
```

---

8. Teste Final (A Integração)

1. Rode o servidor (python manage.py runserver).
2. Acesse o Admin (/admin).
3. Vá em "Projects" e crie 2 ou 3 projetos (ex: "Trabalho", "Estudos", "Casa"). Certifique-se de atribuí-los ao seu usuário admin.
4. **Teste de Segurança (Opcional):** Crie um *outro* superusuário (admin2). Crie um projeto para *ele*.
5. Acesse a página principal (/) (logado como admin).
6. No formulário "Nova Tarefa:", clique no menu suspenso "Project".
7. Você verá **apenas** os projetos que você criou ("Trabalho", "Estudos", "Casa"). Você *não* verá o projeto do admin2. (Isso prova que nosso filtro no TarefaForm funcionou!).
8. Crie uma nova tarefa e associe-a a um projeto.
9. A tarefa deve aparecer na lista, e o nome do projeto (ex: "Projeto: Estudos") deve ser exibido corretamente.

---

9. Revisão da Aula e Conclusão da Série

O que fizemos nesta apostila:

1. **Arquitetura:** Aprendemos *por que* e *como* separar responsabilidades em diferentes apps (startapp projects).
2. **Relações (Inter-App):** Criamos um ForeignKey que liga um Model de um app (core.Tarefa) a um Model de outro app (projects.Project).
3. **Migrations (Multi-App):** Gerenciamos migrações para múltiplos apps simultaneamente.
4. **Forms Avançados:** Aprendemos a sobrescrever o __init__ de um ModelForm para filtrar dinamicamente o queryset de um campo (Project.objects.filter(user=user)).
5. **Views (Integração):** Passamos o request.user da view para o form para que o filtro pudesse funcionar.

O Fim da Jornada (Por Enquanto)

Esta foi nossa última apostila, e foi a mais densa. Olhe para trás e veja o que você construiu.

Começamos na **Apostila 1** com uma única linha (HttpResponse("Olá Mundo")).

Ao longo desta série, você dominou:

- **O Fluxo MVT:** Views, URLs e o básico de manage.py.
- **Templates:** render(), {% tags %}, {{ variaveis }} e Herança ({% extends %}).
- **Banco de Dados:** Models, CharField, BooleanField, makemigrations e migrate.

- **O Admin:** admin.site.register() e como personalizá-lo com list_display, list_filter e search_fields.
- **Formulários:** ModelForm, form.is_valid(), form.save() e o padrão Post-Redirect-Get.
- **CRUD Completo:** Lógica de Create, Read, Update e Delete (.delete(), .save()).
- **Autenticação:** User, LoginView, register, @login_required e request.user.
- **Segurança:** Proteção de dados por usuário (filter(user=request.user)) e {% csrf_token %}.
- **Arquitetura:** E, finalmente, como estruturar um projeto escalável com múltiplos apps.