Tutorial Completo – Iniciando um Projeto Django Organizado (HealthTrack)

Este tutorial mostra **passo a passo** como iniciar um projeto Django do zero, organizando o ambiente, criando apps, modelos, admin, configurações básicas e uma API simples usando Django REST Framework. O objetivo é servir como **material de estudo para outras pessoas**.

1️⃣ Criando a pasta do projeto

Crie uma pasta para o projeto e entre nela:

```
HealthTrack/
```

Essa pasta será a raiz do projeto.

2️⃣ Criando e ativando o ambiente virtual (venv)

Criar o ambiente virtual

```
python -m venv venv
```

- Cria um ambiente Python isolado - Evita conflitos de bibliotecas entre projetos

Ativar a venv (Windows)

```
venv/Scripts/activate
```

Quando ativado, o terminal exibirá `(venv)`.

3️⃣ Atualizando o pip

```
python -m pip install --upgrade pip
```

- Atualiza o gerenciador de pacotes - Evita erros de compatibilidade

---

4️⃣ Instalando dependências do projeto

Django (framework principal)

```
pip install django
```

Requests (requisições HTTP)

```
pip install requests
```

Usado para consumir APIs externas (ex: ViaCEP).

xhtml2pdf (geração de PDFs)

```
pip install xhtml2pdf
```

Permite gerar arquivos PDF a partir de HTML.

Django REST Framework (APIs REST)

```
pip install djangorestframework
```

Permite criar APIs RESTful usando Django.

5️⃣ Comandos básicos de navegação no terminal

- `cd` → navega entre pastas
- `dir` (Windows) / `ls` (Linux/Mac) → lista arquivos do diretório atual

---

6️⃣ Criando o projeto Django

```
django-admin startproject config .
```

Explicação

- `config` é o nome do projeto (settings, urls, wsgi, asgi)
- `.` cria o projeto na pasta atual (boa prática)

Estrutura inicial:

```
config/
manage.py
```

7️⃣ Criando as aplicações (apps)

App principal (core)

```
python manage.py startapp core
```

App de atendimentos

```
python manage.py startapp atendimentos
```

Cada app representa um **domínio do sistema**.

8️⃣ Registrando apps no Django

No arquivo `config/settings.py`, adicione:

```
INSTALLED_APPS = [
    ...
    'core',
    'atendimentos',
    'rest_framework',
]
```

9️⃣ Configurações de sessão

No `settings.py`:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.db'
SESSION_COOKIE_AGE = 1800  # 30 minutos
```

Define onde as sessões são armazenadas e seu tempo de validade.

🔟 Modelos – App Core

Arquivo: `core/models.py`

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

class Paciente(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    cep = models.CharField(max_length=8, blank=True)
    endereco = models.CharField(max_length=100, blank=True)

    def __str__(self):
        return self.user.username
```

Observação

- `blank=True` permite campo vazio no formulário
- O CEP **não é único** (corrigido)

1️⃣1️⃣ Modelos – App Atendimentos

Arquivo: `atendimentos/models.py`

```
from django.db import models
from django.contrib.auth.models import User
from core.models import Paciente

class Especialidade(models.Model):
    nome = models.CharField(max_length=50)

    def __str__(self):
        return self.nome

class Medico(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    especialidade = models.ForeignKey(Especialidade,
on_delete=models.CASCADE)
    crm = models.CharField(max_length=20)

    def __str__(self):
        return self.user.username

class Prontuario(models.Model):
    paciente = models.ForeignKey(Paciente, on_delete=models.CASCADE)
    medico = models.ForeignKey(Medico, on_delete=models.CASCADE)
    data = models.DateTimeField(auto_now_add=True)
    receita = models.TextField()

    def __str__(self):
        return f'Consulta {self.id} - {self.paciente}'
```

1️⃣2️⃣ Criando e aplicando migrações

```
python manage.py makemigrations
python manage.py migrate
```

- `makemigrations` cria os arquivos de migração
- `migrate` aplica no banco de dados

1️⃣3️⃣ Configurando sinais e grupos (core/apps.py)

⚠️ **Correção importante**: o campo correto é `name`, não `nome`.

```
from django.apps import AppConfig
from django.db.models.signals import post_migrate

class CoreConfig(AppConfig):
    name = 'core'
    default_auto_field = 'django.db.models.BigAutoField'

    def ready(self):
        post_migrate.connect(criar_grupos_iniciais, sender=self)

def criar_grupos_iniciais(sender, **kwargs):
    from django.contrib.auth.models import Group
    Group.objects.get_or_create(name='Paciente')
    Group.objects.get_or_create(name='Medico')
```

1️⃣4️⃣ Registrando modelos no Admin

`core/admin.py`

```
from django.contrib import admin
from .models import Paciente

@admin.register(Paciente)
class PacienteAdmin(admin.ModelAdmin):
    list_display = ('user', 'cep', 'endereco')
```

`atendimentos/admin.py`

⚠️ **Correção**: o campo correto é `receita`, não `descricao`.

```
from django.contrib import admin
from .models import Especialidade, Medico, Prontuario
from django.db import models

@admin.register(Medico)
class MedicoAdmin(admin.ModelAdmin):
    list_display = ('user', 'crm', 'especialidade')

admin.site.register(Especialidade)

@admin.register(Prontuario)
class ProntuarioAdmin(admin.ModelAdmin):
    list_display = ('paciente', 'medico', 'data', 'receita')

    actions = ['adicionar_observacao_padrao']

    @admin.action(description='Adicionar observação padrão à receita')
    def adicionar_observacao_padrao(self, request, queryset):
        update = queryset.update(
            receita=models.F('receita') + '\n\nRetorno sugerido em 30 dias.'
        )
        self.message_user(request, f'{update} prontuários atualizados com
sucesso.')
```

1️⃣5️⃣ Criando superusuário

```
python manage.py createsuperuser
```

Permite acessar o Django Admin.

1️⃣6️⃣ Serializers – Cadastro de Paciente (API)

Arquivo: `core/serializers.py`

⚠️ **Correções aplicadas**: - Uso correto de `validated_data[...]` - Campo `user` com letra minúscula

```
from rest_framework import serializers
from django.contrib.auth.models import User, Group
import requests
from .models import Paciente

class CadastroPacienteSerializer(serializers.ModelSerializer):
    username = serializers.CharField(write_only=True)
```

```
    password = serializers.CharField(write_only=True)
    cep = serializers.CharField(max_length=9)

    class Meta:
        model = Paciente
        fields = ['username', 'password', 'cep', 'endereco']

    def validate_cep(self, value):
        cep_limpo = value.replace('-', '')
        url = f'https://viacep.com.br/ws/{cep_limpo}/json/'
        response = requests.get(url)

        if response.status_code != 200 or response.json().get('erro'):
            raise serializers.ValidationError('CEP inválido.')

        self.context['endereco_api'] = response.json().get('logradouro', 'N/
A')
        return value

    def create(self, validated_data):
        username = validated_data.pop('username')
        password = validated_data.pop('password')

        user = User.objects.create_user(username=username, password=password)
        group = Group.objects.get(name='Paciente')
        user.groups.add(group)

        validated_data['endereco'] = self.context['endereco_api']
        paciente = Paciente.objects.create(user=user, **validated_data)
        return paciente
```

✅Conclusão

Esse fluxo cria: - Ambiente profissional - Estrutura modular - Modelos organizados - Admin funcional - Base para API REST

Esse é um **padrão real de projeto Django**, pronto para evoluir para autenticação, permissões, APIs e frontend.