

Apostila Completa: Python Backend + AI/DS

Índice

1. Fundamentos de Python e Git
 2. Bancos de Dados e Django Básico
 3. Python Avançado e Estruturas de Dados
 4. Django Avançado, FastAPI e DevOps
 5. Ciência de Dados e Machine Learning
 6. Projetos Práticos e Preparação para Entrevistas
 7. Temas Avançados para Desenvolvedor Júnior
 8. Recursos e Referências
-

1. Fundamentos de Python e Git {#fundamentos-python-git}

1.1 Sintaxe Básica de Python

Tipos de Dados Básicos

```
python

# Números
inteiro = 42
decimal = 3.14
complexo = 2 + 3j

# Strings
texto = "Hello, World!"
texto_multilinhas = """
Texto com
múltiplas linhas
"""

# Booleanos
verdadeiro = True
falso = False
```

Estruturas de Dados

Listas

```
python
```

```
# Criação e manipulação de listas
```

```
lista = [1, 2, 3, 4, 5]
```

```
lista_mista = [1, "texto", 3.14, True]
```

```
# Métodos essenciais
```

```
lista.append(6)      # Adiciona elemento
```

```
lista.remove(2)      # Remove elemento
```

```
lista.pop()          # Remove último elemento
```

```
lista.insert(0, 0)    # Insere em posição específica
```

Dicionários

```
python
```

```
# Criação e manipulação
```

```
peessoa = {  
    "nome": "João",  
    "idade": 30,  
    "cidade": "São Paulo"  
}
```

```
# Acesso e modificação
```

```
print(peessoa["nome"])  
peessoa["profissao"] = "Desenvolvedor"  
peessoa.update({"telefone": "123456789"})
```

Condicionais e Loops

Estruturas Condicionais

```
python
```

```
idade = 18
```

```
if idade >= 18:  
    print("Maior de idade")  
elif idade >= 16:  
    print("Pode votar")  
else:  
    print("Menor de idade")
```

Loops

```
python
```

For loop

```
for i in range(5):  
    print(f"Número: {i}")
```

While loop

```
contador = 0  
while contador < 5:  
    print(contador)  
    contador += 1
```

List comprehension

```
quadrados = [x**2 for x in range(10)]
```

Funções

python

```
def calcular_area_retangulo(largura, altura):  
    """Calcula a área de um retângulo"""  
    return largura * altura
```

```
def saudacao(nome, sobrenome=""):  
    """Função com parâmetro opcional"""  
    if sobrenome:  
        return f"Olá, {nome} {sobrenome}!"  
    return f"Olá, {nome}!"
```

Função lambda

```
quadrado = lambda x: x**2
```

1.2 Controle de Versão com Git

Comandos Básicos

bash

Configuração inicial

`git config --global user.name "Seu Nome"`

`git config --global user.email "seu.email@exemplo.com"`

Inicializando repositório

`git init`

`git clone https://github.com/usuario/repositorio.git`

Comandos de trabalho diário

`git add .` *# Adiciona arquivos ao staging*

`git commit -m "Mensagem"` *# Confirma alterações*

`git push origin main` *# Envia para repositório remoto*

`git pull origin main` *# Baixa alterações do remoto*

Gerenciamento de branches

`git branch nova-feature` *# Cria nova branch*

`git checkout nova-feature` *# Muda para branch*

`git merge nova-feature` *# Faz merge da branch*

Boas Práticas Git

- Commits pequenos e frequentes
- Mensagens descritivas
- Uso de .gitignore para arquivos desnecessários
- Branching strategy (Git Flow)

2. Bancos de Dados e Django Básico {#bancos-dados-django}

2.1 Fundamentos de Banco de Dados

SQL Básico

sql

-- Criação de tabela

```
CREATE TABLE usuarios (  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  nome VARCHAR(100) NOT NULL,  
  email VARCHAR(100) UNIQUE,  
  idade INTEGER  
);
```

-- Inserção de dados

```
INSERT INTO usuarios (nome, email, idade)  
VALUES ('João Silva', 'joao@email.com', 30);
```

-- Consultas

```
SELECT * FROM usuarios;  
SELECT nome, email FROM usuarios WHERE idade > 25;
```

-- Atualização

```
UPDATE usuarios SET idade = 31 WHERE id = 1;
```

-- Exclusão

```
DELETE FROM usuarios WHERE id = 1;
```

Relacionamentos

sql

-- Tabela de posts

```
CREATE TABLE posts (  
  id INTEGER PRIMARY KEY,  
  titulo VARCHAR(200),  
  conteudo TEXT,  
  usuario_id INTEGER,  
  FOREIGN KEY (usuario_id) REFERENCES usuarios(id)  
);
```

-- Join entre tabelas

```
SELECT u.nome, p.titulo  
FROM usuarios u  
JOIN posts p ON u.id = p.usuario_id;
```

NoSQL com MongoDB

javascript

```
// Inserção de documento
db.usuarios.insertOne({
  nome: "Maria Silva",
  email: "maria@email.com",
  idade: 28,
  hobbies: ["leitura", "natação"]
});

// Consulta
db.usuarios.find({ idade: { $gt: 25 } });

// Atualização
db.usuarios.updateOne(
  { email: "maria@email.com" },
  { $set: { idade: 29 } }
);
```

2.2 Django Básico

Configuração Inicial

```
bash

# Instalação
pip install django

# Criar projeto
django-admin startproject meu_projeto
cd meu_projeto

# Criar app
python manage.py startapp blog

# Migrations
python manage.py makemigrations
python manage.py migrate

# Servidor de desenvolvimento
python manage.py runserver
```

Models (ORM)

```
python
```

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

class Post(models.Model):
    titulo = models.CharField(max_length=200)
    conteudo = models.TextField()
    autor = models.ForeignKey(User, on_delete=models.CASCADE)
    data_criacao = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.titulo

    class Meta:
        ordering = ['-data_criacao']
```

Views

```
python

# views.py
from django.shortcuts import render, get_object_or_404
from django.http import HttpResponse
from .models import Post

def lista_posts(request):
    posts = Post.objects.all()
    return render(request, 'blog/lista.html', {'posts': posts})

def detalhe_post(request, post_id):
    post = get_object_or_404(Post, id=post_id)
    return render(request, 'blog/detalhe.html', {'post': post})

# Class-based views
from django.views.generic import ListView
class PostListView(ListView):
    model = Post
    template_name = 'blog/lista.html'
    context_object_name = 'posts'
```

URLs

```
python
```

```
# urls.py do app
```

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

```
from . import views
```

```
urlpatterns = [
```

```
    path("", views.lista_posts, name='lista_posts'),
```

```
    path('post/<int:post_id>/', views.detalhe_post, name='detalhe_post'),
```

```
]
```

```
# urls.py do projeto
```

```
from django.contrib import admin
```

```
from django.urls import path, include
```

```
urlpatterns = [
```

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

```
    path('blog/', include('blog.urls')),
```

```
]
```

Templates

```
html
```



```
<!-- templates/blog/base.html -->
<!DOCTYPE html>
<html>
<head>
  <title>Meu Blog</title>
</head>
<body>
  <nav>
    <a href="{% url 'lista_posts' %}">Home</a>
  </nav>

  <main>
    {% block content %}
    {% endblock %}
  </main>
</body>
</html>

<!-- templates/blog/lista.html -->
{% extends 'blog/base.html' %}

{% block content %}
  <h1>Posts do Blog</h1>
  {% for post in posts %}
    <article>
      <h2>
        <a href="{% url 'detalhe_post' post.id %}">{{ post.titulo }}</a>
      </h2>
      <p>Por {{ post.autor }} em {{ post.data_criacao }}</p>
    </article>
  {% endfor %}
{% endblock %}
```

3. Python Avançado e Estruturas de Dados {#python-avancado-estruturas}

3.1 Programação Orientada a Objetos

Classes e Objetos

python

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade
        self._cpf = None # Atributo protegido

    def apresentar(self):
        return f"Olá, eu sou {self.nome} e tenho {self.idade} anos"

    def fazer_aniversario(self):
        self.idade += 1

    @property
    def cpf(self):
        return self._cpf

    @cpf.setter
    def cpf(self, valor):
        if len(valor) == 11:
            self._cpf = valor
        else:
            raise ValueError("CPF deve ter 11 dígitos")

# Herança
class Desenvolvedor(Pessoa):
    def __init__(self, nome, idade, linguagem):
        super().__init__(nome, idade)
        self.linguagem = linguagem

    def programar(self):
        return f"{self.nome} está programando em {self.linguagem}"
```

Métodos Especiais

python

```

class ContaBancaria:
    def __init__(self, saldo=0):
        self.saldo = saldo

    def __str__(self):
        return f"Conta com saldo: R$ {self.saldo:.2f}"

    def __add__(self, valor):
        return ContaBancaria(self.saldo + valor)

    def __len__(self):
        return len(str(int(self.saldo)))

# Uso
conta = ContaBancaria(100)
print(conta) # Conta com saldo: R$ 100.00
nova_conta = conta + 50
print(len(conta)) # 3

```

3.2 Estruturas de Dados

Pilha (Stack)

```

python

class Pilha:
    def __init__(self):
        self.items = []

    def empilhar(self, item):
        self.items.append(item)

    def desempilhar(self):
        if not self.vazia():
            return self.items.pop()
        raise IndexError("Pilha vazia")

    def topo(self):
        if not self.vazia():
            return self.items[-1]
        return None

    def vazia(self):
        return len(self.items) == 0

```

Fila (Queue)

python

```
from collections import deque
```

```
class Fila:
```

```
    def __init__(self):
```

```
        self.items = deque()
```

```
    def enfileirar(self, item):
```

```
        self.items.append(item)
```

```
    def desenfileirar(self):
```

```
        if not self.vazia():
```

```
            return self.items.popleft()
```

```
            raise IndexError("Fila vazia")
```

```
    def vazia(self):
```

```
        return len(self.items) == 0
```

Árvore Binária

python

```

class No:
    def __init__(self, valor):
        self.valor = valor
        self.esquerda = None
        self.direita = None

class ArvoreBinaria:
    def __init__(self):
        self.raiz = None

    def inserir(self, valor):
        if self.raiz is None:
            self.raiz = No(valor)
        else:
            self._inserir_recursivo(self.raiz, valor)

    def _inserir_recursivo(self, no, valor):
        if valor < no.valor:
            if no.esquerda is None:
                no.esquerda = No(valor)
            else:
                self._inserir_recursivo(no.esquerda, valor)
        else:
            if no.direita is None:
                no.direita = No(valor)
            else:
                self._inserir_recursivo(no.direita, valor)

```

3.3 Algoritmos de Ordenação

Bubble Sort

```

python

def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
    return lista

```

Quick Sort

```

python

```

```
def quick_sort(lista):  
    if len(lista) <= 1:  
        return lista  
  
    pivot = lista[len(lista) // 2]  
    esquerda = [x for x in lista if x < pivot]  
    meio = [x for x in lista if x == pivot]  
    direita = [x for x in lista if x > pivot]  
  
    return quick_sort(esquerda) + meio + quick_sort(direita)
```

3.4 Complexidade Algorítmica (Big-O)

python

$O(1)$ - Constante

```
def acessar_elemento(lista, indice):  
    return lista[indice]
```

$O(n)$ - Linear

```
def busca_linear(lista, item):  
    for elemento in lista:  
        if elemento == item:  
            return True  
    return False
```

$O(\log n)$ - Logarítmica

```
def busca_binaria(lista, item):  
    inicio = 0  
    fim = len(lista) - 1  
  
    while inicio <= fim:  
        meio = (inicio + fim) // 2  
        if lista[meio] == item:  
            return meio  
        elif lista[meio] < item:  
            inicio = meio + 1  
        else:  
            fim = meio - 1  
  
    return -1
```

$O(n^2)$ - Quadrática

```
def bubble_sort(lista):  
    n = len(lista)  
    for i in range(n):  
        for j in range(0, n - i - 1):  
            if lista[j] > lista[j + 1]:  
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
```

3.5 Padrões de Projeto

Singleton

python

```
class Singleton:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
        return cls._instance
```

Factory

```
python

class AnimalFactory:
    @staticmethod
    def criar_animal(tipo):
        if tipo == "cachorro":
            return Cachorro()
        elif tipo == "gato":
            return Gato()
        else:
            raise ValueError("Tipo de animal não suportado")
```

Observer

```
python

class Observable:
    def __init__(self):
        self._observers = []

    def adicionar_observer(self, observer):
        self._observers.append(observer)

    def notificar_observers(self, *args, **kwargs):
        for observer in self._observers:
            observer.update(*args, **kwargs)
```

4. Django Avançado, FastAPI e DevOps {#django-avancado-fastapi}

4.1 Django REST Framework

Serializers

```
python
```



```
# serializers.py
```

```
from rest_framework import serializers
```

```
from .models import Post
```

```
class PostSerializer(serializers.ModelSerializer):
```

```
    class Meta:
```

```
        model = Post
```

```
        fields = ['id', 'titulo', 'conteudo', 'autor', 'data_criacao']
```

```
        read_only_fields = ['autor', 'data_criacao']
```

ViewSet

```
python
```

```
# views.py
```

```
from rest_framework import viewsets, permissions
```

```
from rest_framework.decorators import action
```

```
from rest_framework.response import Response
```

```
from .models import Post
```

```
from .serializers import PostSerializer
```

```
class PostViewSet(viewsets.ModelViewSet):
```

```
    queryset = Post.objects.all()
```

```
    serializer_class = PostSerializer
```

```
    permission_classes = [permissions.IsAuthenticated]
```

```
    def perform_create(self, serializer):
```

```
        serializer.save(autor=self.request.user)
```

```
    @action(detail=True, methods=['post'])
```

```
    def curtir(self, request, pk=None):
```

```
        post = self.get_object()
```

```
        # Lógica para curtir o post
```

```
        return Response({'status': 'curtido'})
```

Autenticação JWT

```
python
```

```
# settings.py
INSTALLED_APPS = [
    # ...
    'rest_framework',
    'rest_framework_simplejwt',
]

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    )
}

# urls.py
from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
)

urlpatterns = [
    path('api/token/', TokenObtainPairView.as_view()),
    path('api/token/refresh/', TokenRefreshView.as_view()),
]
```

4.2 FastAPI

Aplicação Básica

```
python
```

```
# main.py
```

```
from fastapi import FastAPI, HTTPException, Depends
from pydantic import BaseModel
from typing import List, Optional
import uvicorn
```

```
app = FastAPI(title="Minha API", version="1.0.0")
```

```
# Modelos Pydantic
```

```
class Usuario(BaseModel):
    id: Optional[int] = None
    nome: str
    email: str
    idade: int
```

```
class UsuarioCreate(BaseModel):
    nome: str
    email: str
    idade: int
```

```
# Simulação de banco de dados
```

```
usuarios_db = []
```

```
@app.get("/")
async def root():
    return {"message": "Hello World"}
```

```
@app.get("/usuarios/", response_model=List[Usuario])
async def listar_usuarios():
    return usuarios_db
```

```
@app.get("/usuarios/{user_id}", response_model=Usuario)
async def obter_usuario(user_id: int):
    for usuario in usuarios_db:
        if usuario.id == user_id:
            return usuario
    raise HTTPException(status_code=404, detail="Usuário não encontrado")
```

```
@app.post("/usuarios/", response_model=Usuario)
async def criar_usuario(usuario: UsuarioCreate):
    novo_usuario = Usuario(
        id=len(usuarios_db) + 1,
        **usuario.dict()
    )
```

```
usuarios_db.append(novo_usuario)
return novo_usuario
```

Dependências e Autenticação

```
python

from fastapi import Depends, HTTPException, status
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
import jwt

security = HTTPBearer()

def verificar_token(credentials: HTTPAuthorizationCredentials = Depends(security)):
    try:
        payload = jwt.decode(credentials.credentials, "SECRET_KEY", algorithms=["HS256"])
        return payload
    except jwt.InvalidTokenError:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token inválido"
        )

@app.get("/perfil/")
async def obter_perfil(token_data = Depends(verificar_token)):
    return {"user_id": token_data["user_id"]}
```

4.3 Docker

Dockerfile para Django

```
dockerfile

FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

Docker Compose

```
yaml

# docker-compose.yml
version: '3.8'

services:
  web:
    build: .
    ports:
      - "8000:8000"
    volumes:
      - ./app
    depends_on:
      - db
    environment:
      - DATABASE_URL=postgresql://user:password@db:5432/myapp

  db:
    image: postgres:13
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    environment:
      - POSTGRES_DB=myapp
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=password

volumes:
  postgres_data:
```

4.4 Testes Automatizados

Pytest

```
python
```

```
# test_models.py
import pytest
from django.test import TestCase
from django.contrib.auth.models import User
from blog.models import Post
```

```
@pytest.mark.django_db
class TestPost:
    def test_criar_post(self):
        user = User.objects.create_user(
            username='testuser',
            password='testpass'
        )
        post = Post.objects.create(
            titulo='Teste',
            conteudo='Conteúdo de teste',
            autor=user
        )
        assert post.titulo == 'Teste'
        assert post.autor == user
```

```
# test_api.py
from fastapi.testclient import TestClient
from main import app

client = TestClient(app)

def test_criar_usuario():
    response = client.post(
        "/usuarios/",
        json={"nome": "João", "email": "joao@test.com", "idade": 30}
    )
    assert response.status_code == 200
    assert response.json()["nome"] == "João"
```

5. Ciência de Dados e Machine Learning {#ciencia-dados-ml}

5.1 NumPy

Arrays Multidimensionais

```
python
```

```
import numpy as np
```

```
# Criação de arrays
```

```
arr1d = np.array([1, 2, 3, 4, 5])
```

```
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
```

```
zeros = np.zeros((3, 3))
```

```
ones = np.ones((2, 4))
```

```
identidade = np.eye(3)
```

```
# Operações matemáticas
```

```
a = np.array([1, 2, 3])
```

```
b = np.array([4, 5, 6])
```

```
soma = a + b
```

```
produto = a * b
```

```
produto_escalar = np.dot(a, b)
```

```
# Funções estatísticas
```

```
dados = np.array([1, 2, 3, 4, 5])
```

```
media = np.mean(dados)
```

```
desvio = np.std(dados)
```

```
maximo = np.max(dados)
```

Indexação e Slicing

```
python
```

```
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
# Acessando elementos
```

```
elemento = arr[1, 2] # 6
```

```
linha = arr[1, :] # [4, 5, 6]
```

```
coluna = arr[:, 1] # [2, 5, 8]
```

```
# Condições booleanas
```

```
maiores_que_5 = arr[arr > 5] # [6, 7, 8, 9]
```

5.2 Pandas

DataFrames

```
python
```

```
import pandas as pd

# Criação de DataFrame
dados = {
    'nome': ['Ana', 'Bruno', 'Carlos'],
    'idade': [25, 30, 35],
    'cidade': ['SP', 'RJ', 'BH']
}
df = pd.DataFrame(dados)

# Leitura de arquivos
df_csv = pd.read_csv('dados.csv')
df_excel = pd.read_excel('dados.xlsx')

# Informações básicas
print(df.head())    # Primeiras 5 linhas
print(df.info())    # Informações gerais
print(df.describe()) # Estatísticas descritivas
```

Manipulação de Dados

```
python

# Seleção
nomes = df['nome']
subset = df[['nome', 'idade']]
filtro = df[df['idade'] > 25]

# Adição de colunas
df['salario'] = [5000, 6000, 7000]
df['categoria'] = df['idade'].apply(lambda x: 'jovem' if x < 30 else 'adulto')

# Groupby
agrupado = df.groupby('categoria')['salario'].mean()

# Merge
df2 = pd.DataFrame({
    'nome': ['Ana', 'Bruno'],
    'departamento': ['TI', 'RH']
})
df_merged = pd.merge(df, df2, on='nome', how='left')
```

Limpeza de Dados

```
python
```


Valores ausentes

```
df.isnull().sum()      # Contar valores nulos  
df.dropna()           # Remover linhas com valores nulos  
df.fillna(0)          # Preencher valores nulos
```

Duplicatas

```
df.duplicated().sum()  # Contar duplicatas  
df.drop_duplicates()   # Remover duplicatas
```

Conversão de tipos

```
df['idade'] = df['idade'].astype(int)  
df['data'] = pd.to_datetime(df['data'])
```

5.3 Matplotlib

Gráficos Básicos

python

```
import matplotlib.pyplot as plt

# Gráfico de linha
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.figure(figsize=(8, 6))
plt.plot(x, y, marker='o')
plt.title('Gráfico de Linha')
plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')
plt.grid(True)
plt.show()

# Gráfico de barras
categorias = ['A', 'B', 'C', 'D']
valores = [23, 45, 56, 78]

plt.bar(categorias, valores)
plt.title('Gráfico de Barras')
plt.show()

# Histograma
dados = np.random.normal(0, 1, 1000)
plt.hist(dados, bins=30, alpha=0.7)
plt.title('Histograma')
plt.show()
```

5.4 Scikit-Learn

Regressão Linear

```
python
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
```

```
# Dados de exemplo
```

```
X = np.array([[1], [2], [3], [4], [5]])
```

```
y = np.array([2, 4, 6, 8, 10])
```

```
# Divisão treino/teste
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
# Treinamento
```

```
modelo = LinearRegression()
```

```
modelo.fit(X_train, y_train)
```

```
# Predição
```

```
y_pred = modelo.predict(X_test)
```

```
# Avaliação
```

```
mse = mean_squared_error(y_test, y_pred)
```

```
r2 = r2_score(y_test, y_pred)
```

```
print(f"MSE: {mse}")
```

```
print(f"R2: {r2}")
```

Classificação

```
python
```

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Carregando dados
iris = load_iris()
X, y = iris.data, iris.target

# Divisão treino/teste
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Treinamento
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predição
y_pred = clf.predict(X_test)

# Avaliação
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

Validação Cruzada

```
python

from sklearn.model_selection import cross_val_score

# Validação cruzada
scores = cross_val_score(clf, X, y, cv=5)
print(f"Acurácia média: {scores.mean():.2f} (+/- {scores.std() * 2:.2f})")
```

6. Projetos Práticos e Preparação para Entrevistas {#projetos-entrevistas}

6.1 Projeto: API de Blog com Django REST

```
python
```

```
# models.py
```

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

```
class Categoria(models.Model):
    nome = models.CharField(max_length=100)
    slug = models.SlugField(unique=True)
```

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

```
class Post(models.Model):
    titulo = models.CharField(max_length=200)
    slug = models.SlugField(unique=True)
    conteudo = models.TextField()
    resumo = models.TextField(max_length=300)
    autor = models.ForeignKey(User, on_delete=models.CASCADE)
    categoria = models.ForeignKey(Categoria, on_delete=models.CASCADE)
    publicado = models.BooleanField(default=False)
    data_criacao = models.DateTimeField(auto_now_add=True)
    data_atualizacao = models.DateTimeField(auto_now=True)
```

```
class Meta:
    ordering = ['-data_criacao']
```

```
# serializers.py
```

```
from rest_framework import serializers
from .models import Post, Categoria
```

```
class CategoriaSerializer(serializers.ModelSerializer):
```

```
    class Meta:
        model = Categoria
        fields = '__all__'
```

```
class PostSerializer(serializers.ModelSerializer):
```

```
    autor_nome = serializers.CharField(source='autor.username', read_only=True)
    categoria_nome = serializers.CharField(source='categoria.nome')
```