**User personalizado**

Aqui vamos criar um usuário customizado chamado Usuario, herdando de AbstractUser para manter as funcionalidades padrões e adicionar novos campos.

**Arquivo models.py:**

from django.contrib.auth.models import AbstractUser, BaseUserManager

from django.db import models

# Gerenciador de usuários customizado

class UsuarioManager(BaseUserManager):

def create\_user(self, email, username, password=None, \*\*extra\_fields):

if not email:

raise ValueError('O usuário deve ter um endereço de e-mail')

email = self.normalize\_email(email)

user = self.model(email=email, username=username, \*\*extra\_fields)

user.set\_password(password)

user.save(using=self.\_db)

return user

def create\_superuser(self, email, username, password=None, \*\*extra\_fields):

extra\_fields.setdefault('is\_staff', True)

extra\_fields.setdefault('is\_superuser', True)

return self.create\_user(email, username, password, \*\*extra\_fields)

# Modelo de Usuário

class Usuario(AbstractUser):

email = models.EmailField(unique=True)

telefone = models.CharField(max\_length=15, blank=True, null=True)

data\_nascimento = models.DateField(blank=True, null=True)

# Substituindo o gerenciador padrão

objects = UsuarioManager()

# Campo usado para login (username ou email)

USERNAME\_FIELD = 'email'

REQUIRED\_FIELDS = ['username'] # Campos obrigatórios além do USERNAME\_FIELD

def \_\_str\_\_(self):

return self.email

class Meta:

verbose\_name = 'Usuário'

verbose\_name\_plural = 'Usuários'

Diga ao Django para usar o seu modelo customizado.

**Arquivo settings.py:**

# Substitua app\_nome pelo nome do seu app

AUTH\_USER\_MODEL = 'app\_nome.Usuario'

Para gerenciar no Django Admin.

**Arquivo admin.py:**

from django.contrib import admin

from django.contrib.auth.admin import UserAdmin

from .models import Usuario

@admin.register(Usuario)

class UsuarioAdmin(UserAdmin):

model = Usuario

list\_display = ('email', 'username', 'is\_staff', 'is\_active')

list\_filter = ('is\_staff', 'is\_active')

search\_fields = ('email', 'username')

ordering = ('email',)

fieldsets = (

(None, {'fields': ('email', 'username', 'password')}),

('Permissões', {'fields': ('is\_staff', 'is\_active', 'is\_superuser', 'groups', 'user\_permissions')}),

('Informações Pessoais', {'fields': ('telefone', 'data\_nascimento')}),

)

add\_fieldsets = (

(None, {

'classes': ('wide',),

'fields': ('email', 'username', 'password1', 'password2', 'is\_staff', 'is\_active')}

),

)

Depois de definir o modelo, use get\_user\_model() para buscar o usuário sem importar diretamente.

**Arquivo views.py:**

from django.contrib.auth import get\_user\_model

from django.http import HttpResponse

User = get\_user\_model()

def criar\_usuario(request):

user = User.objects.create\_user(

email='novo@exemplo.com',

username='novousuario',

password='senha123'

)

return HttpResponse(f'Usuário criado: {user.email}')

-----------------------------------------------------------------------------------------------

O **get\_user\_model()** não faz parte da criação do modelo (passos 1–3).

Ele é usado quando você precisa manipular o modelo de usuário em outro lugar do projeto, como:

**- Views (funções ou class-based views)**

**- Serializers (se estiver usando DRF)**

**- Forms**

**- Scripts ou testes automatizados**

Ou seja, ele aparece quando você quer interagir com o usuário, e não dentro do modelo ou admin.