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.