Modelos de Classe do AlbumFlow

Classe referente a Album:

```python
from django.db import models

from user.models import Photographer, Tag
from gallery.utils import generate_custom_id


class Album(models.Model):
    """
    Representa um álbum de fotos associado a um fotógrafo.

    Atributos:
        name_album (str): Nome do álbum (máximo de 50 caracteres).
        photographer (Photographer): Fotógrafo proprietário do álbum.
        tags (ManyToManyField): Lista de tags associadas ao álbum
(opcional).
        s3_path_album (str): Caminho onde o álbum está armazenado no AWS S3.
        created_at_album (datetime): Data e hora da criação do álbum.
        updated_at_album (datetime): Data e hora da última atualização do
álbum.
    """
    name_album = models.CharField(
        max_length=50,
        verbose_name="Album name"
    )

    photographer = models.ForeignKey(
        Photographer,
        on_delete=models.CASCADE,
        verbose_name="Photographer"
    )

    tags = models.ManyToManyField(
        Tag,
        verbose_name="Album tag list",
        null=True,
        blank=True
    )

    s3_path_album = models.CharField(
        max_length=255,
        blank=True,
        editable=False,
        verbose_name="Album path S3"
    )
```

```
    created_at_album = models.DateTimeField(
        auto_now_add=True,
        verbose_name="Album creation date"
    )

    updated_at_album = models.DateTimeField(
        auto_now=True,
        verbose_name="Album update date"
    )

    def __str__(self):
        """
        Retorna uma representação em string do álbum.

        Retorna:
            str: Uma string contendo o ID e o nome do álbum.
        """
        return f"{self.id} - {self.name_album}"

    def save(self, *args, **kwargs):
        """
        Sobrescreve o método save para gerar o caminho no S3 para o álbum,
caso ele não exista.
        O caminho é construído usando a pasta do fotógrafo no S3 e um ID
personalizado gerado.

        Args:
            *args: Argumentos posicionais adicionais.
            **kwargs: Argumentos nomeados adicionais.
        """
        if not self.s3_path_album:
            self.s3_path_album =
f"{self.photographer.s3_folder}albums/{generate_custom_id(self.name_album)}/
"

        super().save(*args, **kwargs)

    class Meta:
        """
        Configurações de metadados para o modelo Album.
        """
        verbose_name = "Album",
        verbose_name_plural = "Albuns"
        ordering = ['id', 'name_album']
```

Classe referente a Foto:

Python

```python
from django.db import models
from storages.backends.s3boto3 import S3Boto3Storage
import os
from uuid import uuid4
from django.core.files.storage import default_storage
from django.utils.text import slugify

from .album import Album
from gallery.utils import generate_custom_id

class PhotoStorage(S3Boto3Storage):
    """
    Configuração personalizada de armazenamento para fotos utilizando Amazon S3.

    Atributos:
        file_overwrite (bool): Define se arquivos com o mesmo nome podem ser sobrescritos (False).
        default_acl (str): Define o controle de acesso padrão dos arquivos (privado).

    Métodos:
        get_valid_name(name): Normaliza o nome do arquivo, garantindo um caminho correto.
        generate_filename(filename): Normaliza o nome do arquivo gerado, evitando problemas de caminho.
    """
    file_overwrite = False
    default_acl = 'private'

    def get_valid_name(self, name):
        """
        Normaliza o nome do arquivo removendo barras invertidas e garantindo um caminho correto.

        Args:
            name (str): Nome original do arquivo.

        Returns:
            str: Nome normalizado do arquivo.
        """
        name = name.replace('\\', '/')
        name = os.path.normpath(name).replace('\\', '/')

        return super().get_valid_name(name)
```

```
    def generate_filename(self, filename):
        """
        Normaliza o nome do arquivo antes de salvá-lo.

        Args:
            filename (str): Nome original do arquivo.

        Returns:
            str: Nome normalizado do arquivo.
        """
        filename = filename.replace('\\', '/')
        filename = os.path.normpath(filename).replace('\\', '/')

        return super().generate_filename(filename)

class Photo(models.Model):
    """
    Representa uma foto associada a um álbum, armazenada no Amazon S3.

    Atributos:
        album (Album): Referência ao álbum ao qual a foto pertence.
        photo (FileField): Arquivo da foto armazenado no S3.
        s3_path_photo (str): Caminho da foto dentro do S3.

    Propriedades:
        name_photo (str): Retorna o nome do arquivo da foto.
        format_photo (str): Retorna o formato da foto (extensão do arquivo).
        size_photo (int): Retorna o tamanho do arquivo da foto.

    Métodos:
        clean_path(path): Normaliza o caminho da foto removendo barras
duplicadas.
        save(*args, **kwargs): Gera o caminho correto para a foto no S3
antes de salvá-la.
    """
    album = models.ForeignKey(
        Album,
        on_delete=models.CASCADE,
        verbose_name="Photo album"
    )

    photo = models.FileField(
        storage=PhotoStorage(),
        upload_to='',
        max_length=500,
        verbose_name="Photo file"
    )
```

```
    s3_path_photo = models.CharField(
        max_length=500,
        blank=True,
        editable=False,
        verbose_name="Photo path S3"
    )

    @property
    def name_photo(self):
        """
        Retorna o nome do arquivo da foto.

        Returns:
            str: Nome do arquivo da foto.
        """
        return self.photo.name

    @property
    def format_photo(self):
        """
        Retorna o formato da foto (extensão do arquivo).

        Returns:
            str: Formato da foto em letras maiúsculas (exemplo: JPG, PNG).
        """
        return os.path.splitext(self.photo.name)[1][1:].upper() if
self.photo else ''

    @property
    def size_photo(self):
        """
        Retorna o tamanho do arquivo da foto em bytes.

        Returns:
            int: Tamanho do arquivo da foto.
        """
        return self.photo.size if self.photo else 0

    def clean_path(self, path):
        """
        Normaliza o caminho do arquivo removendo barras invertidas e barras
duplas.

        Args:
            path (str): Caminho original.

        Returns:
```

```
            str: Caminho normalizado.
        """
        path = os.path.normpath(path).replace('\\', '/')

        return path.replace('//', '/')

    def save(self, *args, **kwargs):
        """
        Gera o caminho correto para a foto no S3 antes de salvá-la.

        Se o caminho `s3_path_photo` ainda não foi definido, ele será gerado
automaticamente
        baseado no álbum associado.

        Args:
            *args: Argumentos posicionais adicionais.
            **kwargs: Argumentos nomeados adicionais.
        """
        if not self.s3_path_photo and hasattr(self, 'album') and
self.album.s3_path_album:
            clean_album_path = self.clean_path(self.album.s3_path_album)

            if not clean_album_path.endswith('/'):
                clean_album_path += '/'

            ext = os.path.splitext(self.photo.name)[1] if self.photo else
'.jpg'
            unique_filename = f"{generate_custom_id(self.name_photo)}{ext}"

            self.s3_path_photo =
self.clean_path(f"{clean_album_path}{unique_filename}")

            if self.photo:
                self.photo.name = self.s3_path_photo

        super().save(*args, **kwargs)

    def __str__(self):
        """
        Retorna uma representação em string da foto.

        Returns:
            str: Uma string contendo o ID e o nome da foto.
        """
        return f"{self.id} - {self.name_photo}"

    class Meta:
        """
```

```
        Configurações de metadados para o modelo Photo.
        """
        verbose_name = "Photo"
        verbose_name_plural = "Photos"
        ordering = ['id']
```

Classe referente a Fotografo:

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

from gallery.utils import generate_custom_id

class Profile(models.Model):
    """
    Representa o perfil de um usuário no sistema.

    Atributos:
        user (User): Relacionamento um para um com o modelo User,
representando o usuário associado ao perfil.
        type_user (str): Tipo do usuário (máximo de 50 caracteres).
        date_created (datetime): Data de criação do perfil.
        date_updated (datetime): Data da última atualização do perfil.
    """
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    type_user = models.CharField(max_length=50)
    date_created = models.DateField(auto_now_add=True)
    date_updated = models.DateField(auto_now=True)

    class Meta:
        """
        Configurações de metadados para a classe Profile.
        """
        abstract = True

class Photographer(Profile):
    """
    Representa um fotógrafo no sistema, herdando de Profile.

    Atributos:
        name_photographer (str): Nome do fotógrafo (máximo de 60
caracteres).
        name_company (str): Nome da empresa do fotógrafo (máximo de 50
caracteres).
        phone (str): Número de telefone do fotógrafo (máximo de 20
caracteres).
        s3_folder (str): Diretório no S3 onde as imagens do fotógrafo são
armazenadas (padrão gerado automaticamente).
    """
    name_photographer = models.CharField(max_length=60)
    name_company = models.CharField(max_length=50)
    phone = models.CharField(max_length=20)
```

```
    s3_folder = models.CharField(max_length=255, blank=True, editable=False)

    def save(self, *args, **kwargs):
        """
        Sobrescreve o método save para gerar automaticamente o diretório no
S3, caso ele não exista.
        O diretório é construído usando um ID personalizado gerado a partir
do nome do fotógrafo.

        Args:
            *args: Argumentos posicionais adicionais.
            **kwargs: Argumentos nomeados adicionais.
        """
        if not self.s3_folder:
            self.s3_folder =
f"{generate_custom_id(f"{self.name_photographer}")}/"

        super().save(*args, **kwargs)

    def __str__(self):
        """
        Retorna uma representação em string do fotógrafo.

        Retorna:
            str: Uma string contendo o tipo de usuário e o nome do
fotógrafo.
        """
        return f"{self.type_user} - {self.name_photographer}"

    class Meta:
        """
        Configurações de metadados para o modelo Photographer.
        """
        verbose_name = "Photographer"
        verbose_name_plural = "Photographers"
```

Classe referente a Tag:

Python

```python
from django.db import models

from user.models import Photographer


class Tag(models.Model):
    """
    Representa uma tag associada a um fotógrafo.

    Atributos:
        name_tag (str): Nome da tag (máximo de 20 caracteres).
        photographer (Photographer): Fotógrafo ao qual a tag está associada.
    """
    name_tag = models.CharField(
        max_length=20,
        verbose_name="Tag name"
    )

    photographer = models.ForeignKey(
        Photographer,
        on_delete=models.CASCADE,
        verbose_name="Photographer"
    )

    def __str__(self):
        """
        Retorna uma representação em string da tag.

        Retorna:
            str: Uma string contendo o ID e o nome da tag.
        """
        return f"{self.id} - {self.name_tag}"

    class Meta:
        """
        Configurações de metadados para o modelo Tag.
        """
        verbose_name = "Tag",
        verbose_name_plural = "Tags"
        ordering = ['id', 'name_tag']
```