Universidad Autónoma de Chihuahua

PROYECTO SEGUNDO PARCIAL

Blog de Recetas Con Mongo

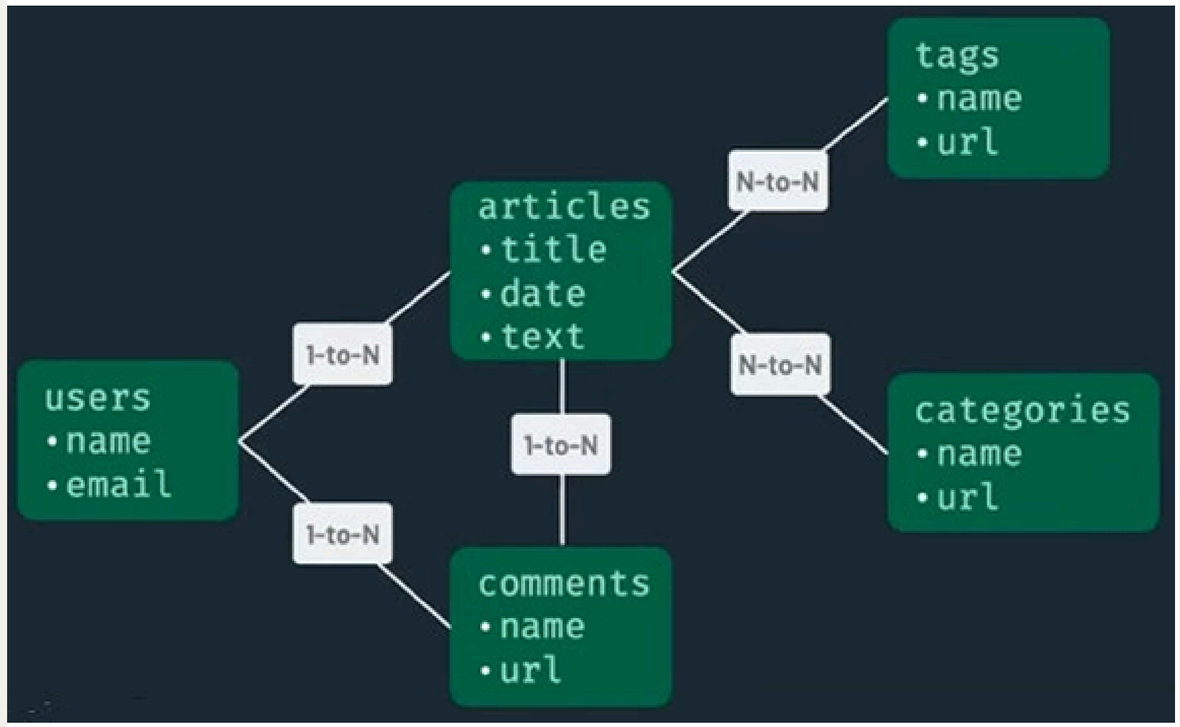
Joel Omar Castillo Castillo #
Paloma Ivonne Sandoval Granados #368027
Idaly Guadalupe Morales Robredo # 367926

BASE EN MONGO

En esta base de datos cumplimos con las relaciones definidas en el diagrama utilizando un sistema de “Referencias” que simula los JOINs de SQL mediante $lookup.
Guardamos el _id del documento relacionado

```
_id: ObjectId('690e68f8fc47033e857558ae')
title : "Receta de pastel de chocolate casero"
date : 2025-11-07T00:00:00.000+00:00
text : "receta muy rica de pastel con un sabor casero"
user_id : ObjectId('690e67d3fc47033e857558a8')
▾ categories : Array (1)
    0: ObjectId('690e6840fc47033e857558ac')
▾ tags : Array (1)
    0: ObjectId('690e6855fc47033e857558ad')
```

En la colección articles el campo de user guarda el id del autor que esta en la colección users



RELACIONES

- Todas las relaciones se cumplen mediante las referencias de _id donde un documento apunta al id del otro

UNO A MUCHOS

nuestro usuario puede ser el autor de muchos artículos pero el articulo tiene un solo autor. el documento del articulo almacena el id del usuario

```
_id: ObjectId('690e68f8fc47033e857558ae')
title : "Receta de pastel de chocolate casero"
date : 2025-11-07T00:00:00.000+00:00
text : "receta muy rica de pastel con un sabor casero"
user_id : ObjectId('690e67d3fc47033e857558a8')
```

```
_id: ObjectId('690e67d3fc47033e857558a8')
name : "pepe pecas"
email : "pepepecas@gmail.com"
password : "123"
```

- la relación que hay entre articles y comments es básicamente que un articulo puede recibir varios comentarios, pero cada comentario esta ligado a un solo articulo

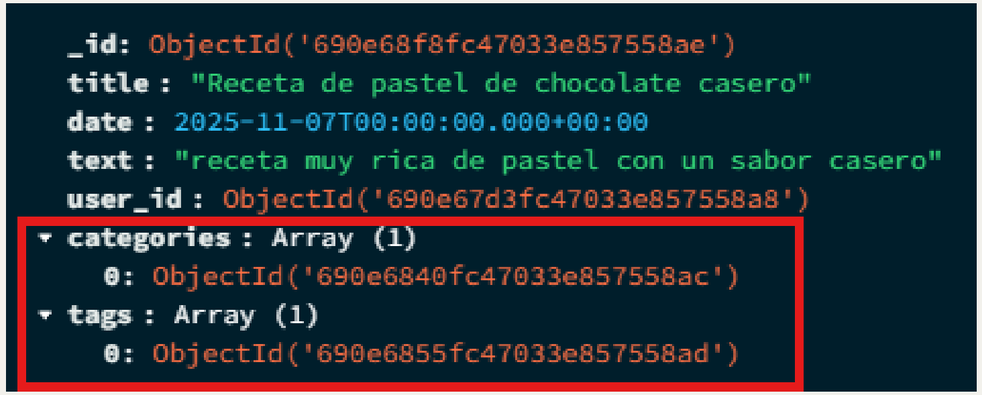
```
_id: ObjectId('690e6973fc47033e857558b0')
name : "pepe pecas"
text : "esta muy bueno yo lo hice para mis hijos"
article_id : ObjectId('690e68f8fc47033e857558ae')
user_id : ObjectId('690e67d3fc47033e857558a8')
```

```
_id: ObjectId('690e68f8fc47033e857558ae')
title : "Receta de pastel de chocolate casero"
date : 2025-11-07T00:00:00.000+00:00
text : "receta muy rica de pastel con un sabor casero"
user_id : ObjectId('690e67d3fc47033e857558a8')
▾ categories : Array (1)
    0: ObjectId('690e6840fc47033e857558ac')
▾ tags : Array (1)
    0: ObjectId('690e6855fc47033e857558ad')
```

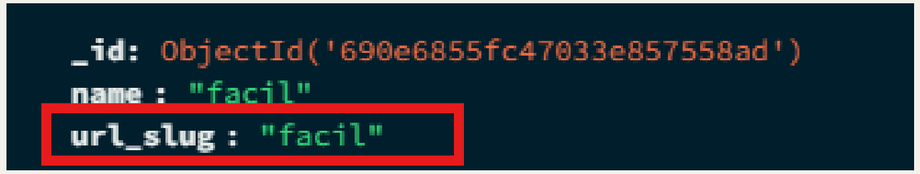
RELACIONES

MUCHOS A MUCHOS

en la relación artículos con tags, un articulo puede tener múltiples tags y una tag puede aplicarse a múltiples artículos



usamos un array para los tags para asi permitir que almacene multiples referencias



creamos un id_slug que es un id aparte del que ya genera mongo automáticamente, mas que nada por practicidad ya que perfectamente se puede manejar usando la que entrega mongo. El id slug nos serviría para construir un Url mas limpio mostrando tipo:

https://Blog.com/tags/facil

CONEXION

Para hacer la conexión con nuestra base usamos pymongo `from pymongo import MongoClient` después de eso creamos la instancia del cliente tratando de hacer conexión con el host que hicimos, luego seleccionamos la base de datos que vamos a utilizar

```python
# --- Configuración de la Conexión ---
#aqui definimos los parametros de nuestra base de mongo
MONGO_URL = "mongodb://localhost:27017/"
DB_NAME = "Blog_Receta"

class ConexionMongoDB:
    """Clase para gestionar la conexión a MongoDB y el objeto de la base de datos."""

    def __init__(self, raiz=None):
        """ Inicializa la conexión."""
        self.cliente = None # Inicializa el cliente de MongoDB a None.
        self.db = None      # Inicializa el objeto de la base de datos a None.

        try:
            # Crea una instancia de MongoClient intentando conectarse a la URL definida
            self.cliente = MongoClient(MONGO_URL)
            # Prueba la conexión con el comando 'ping' (petición rápida al servidor).
            self.cliente.admin.command('ping')
            # Si el ping es exitoso, selecciona la base de datos específica.
            self.db = self.cliente[DB_NAME]
            print("¡Conexión a MongoDB exitosa!")
```

CONEXION

esta parte nos vincula con los demás códigos porque le dice a la variable DB que guarde el objeto de la base que agarramos y esto hace que funcione para que los demás códigos puedan acceder. un ejemplo es en la parte de lógica cuando importamos DB

```
#instancia global
# Este bloque garantiza que la conexion se intente una vez al cargar el mo
try:
    # Intenta crear la conexión.
    gestor_db = ConexionMongoDB()
    # Almacena el objeto de la base de datos en la variable global DB
    DB = gestor_db.obtener_db()
except:
    # en caso de la conexión falla se establece a None.
    # Esto es manejado por los gestores en Logica.py.
    DB = None
```

VERIFICAR USUARIO (LOGICA.PY)

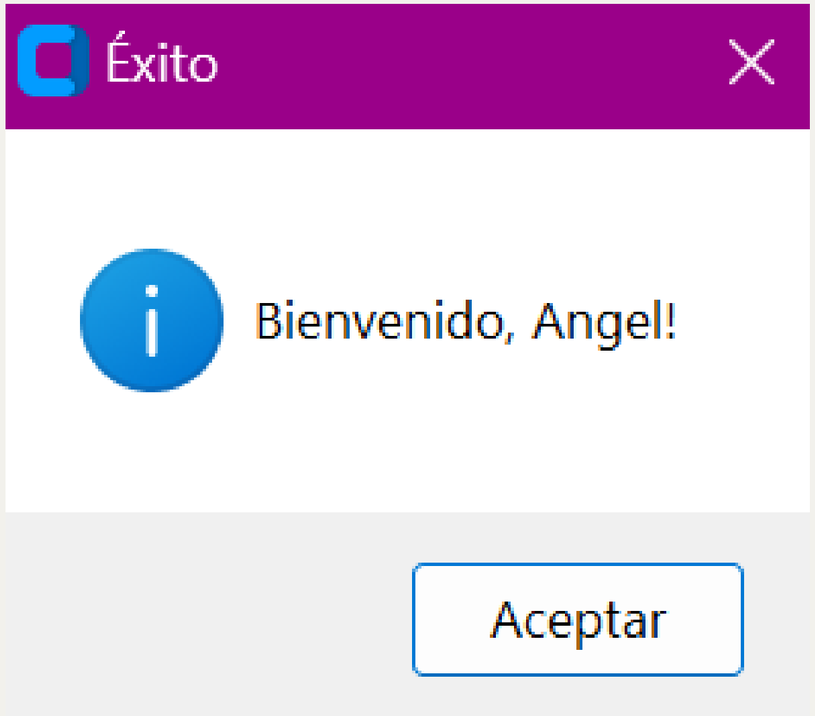
Se realiza una consulta self.coleccion.find_one({"email": email, "password": password}). Esta consulta a la colección users busca un documento que coincida exactamente con el email y la contraseña proporcionados.

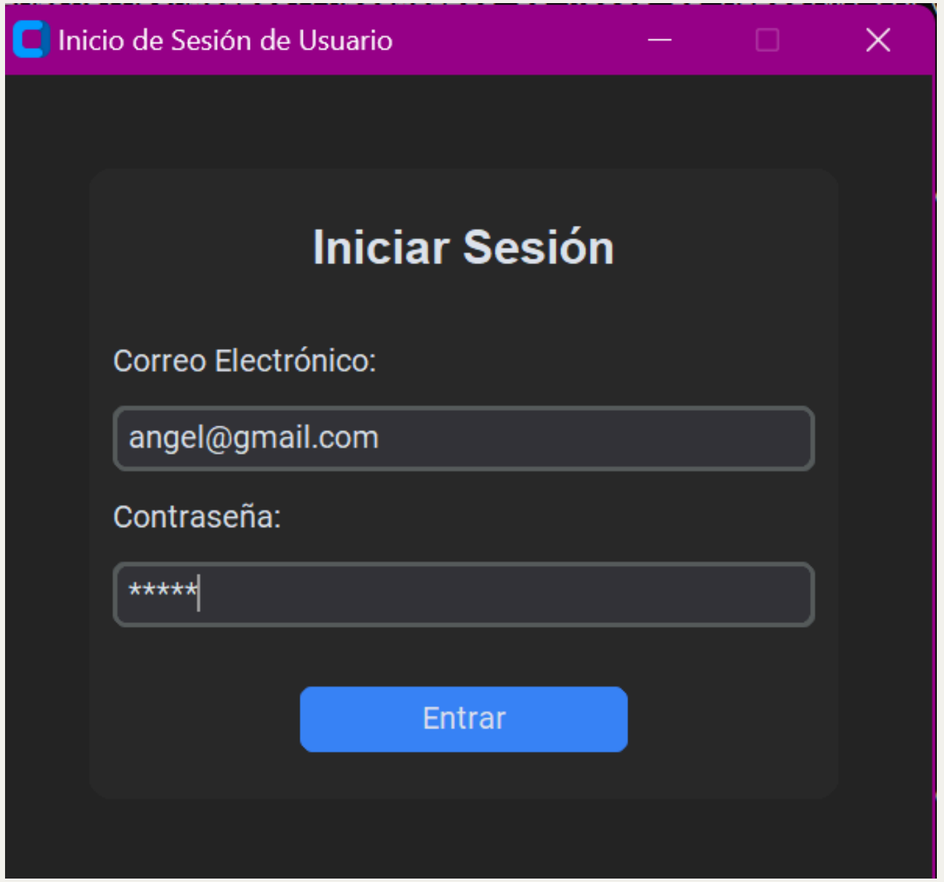
```python
class GestorUsuario(GestorEntidad):
    def __init__(self):
        # Llama al constructor de la clase base, usando "email" como clave_nombre.
        super().__init__("users", clave_nombre="email")

    def autenticar(self, email, password):
        """Verifica el email y la contraseña contra la base de datos."""
        try:
            # Busca un documento que coincida con el email Y la contraseña.
            # NOTA: En una app real, la contraseña debe estar hasheada (no en texto plano).
            usuario = self.coleccion.find_one({"email": email, "password": password})
            return usuario # Retorna el documento del usuario si la autenticación es exitosa, s
        except Exception as e:
            print(f"Error de autenticación: {e}")
            return None

```

RESULTADOS


Éxito
Bienvenido, Angel!
Aceptar


Inicio de Sesión de Usuario
Iniciar Sesión
Correo Electrónico:
angel@gmail.com
Contraseña:
*****
Entrar

CRUD GENÉRICAS (LOGICA.PY)

Insertar

```python
def crear_uno(self, datos):
    """Inserta un nuevo documento en la colección."""
    try:
        resultado = self.coleccion.insert_one(datos)
        self.cargar_mapa() # El mapa debe recargarse para incluir el nuevo elemento.
        return resultado.inserted_id # Retorna el ID generado por MongoDB.
    except Exception as e:
        print(f"Error creando el documento en {self.coleccion.name}: {e}")
        return None
```

Llama a self.coleccion.insert_one(datos). Esto inserta un nuevo documento. Importante: Después de la inserción, llama a self.cargar_mapa() para que el mapa cache (mapa_nombre_a_id) esté actualizado.

Buscar

```python
def obtener_todos(self):
    """Retorna todos los documentos de la colección como una lista."""
    try:
        return list(self.coleccion.find()) # Realiza la consulta a la base de datos.
    except Exception as e:
        print(f"Error obteniendo todos los documentos para {self.coleccion.name}: {e}")
        return []
```

Llama a list(self.coleccion.find()) para retornar todos los documentos de la colección.

CRUD GENÉRICAS (LOGICA.PY)

Actualizar

```python
def actualizar_uno(self, id_objeto, nuevos_datos):
    """Actualiza un documento por su ObjectId."""
    try:
        # Usa $set para actualizar solo los campos en 'nuevos_datos'.
        resultado = self.coleccion.update_one({"_id": id_objeto}, {"$set": nuevos_datos})
        self.cargar_mapa() # Recarga el mapa por si el nombre/email fue actualizado.
        return resultado.modified_count # Retorna cuántos documentos fueron modificados (
    except Exception as e:
        print(f"Error actualizando el documento en {self.coleccion.name}: {e}")
        return 0
```

Llama a self.coleccion.update_one({"_id": id_objeto}, {"$set": nuevos_datos}). El operador $set asegura que solo se modifiquen los campos incluidos en nuevos_datos.

Eliminar

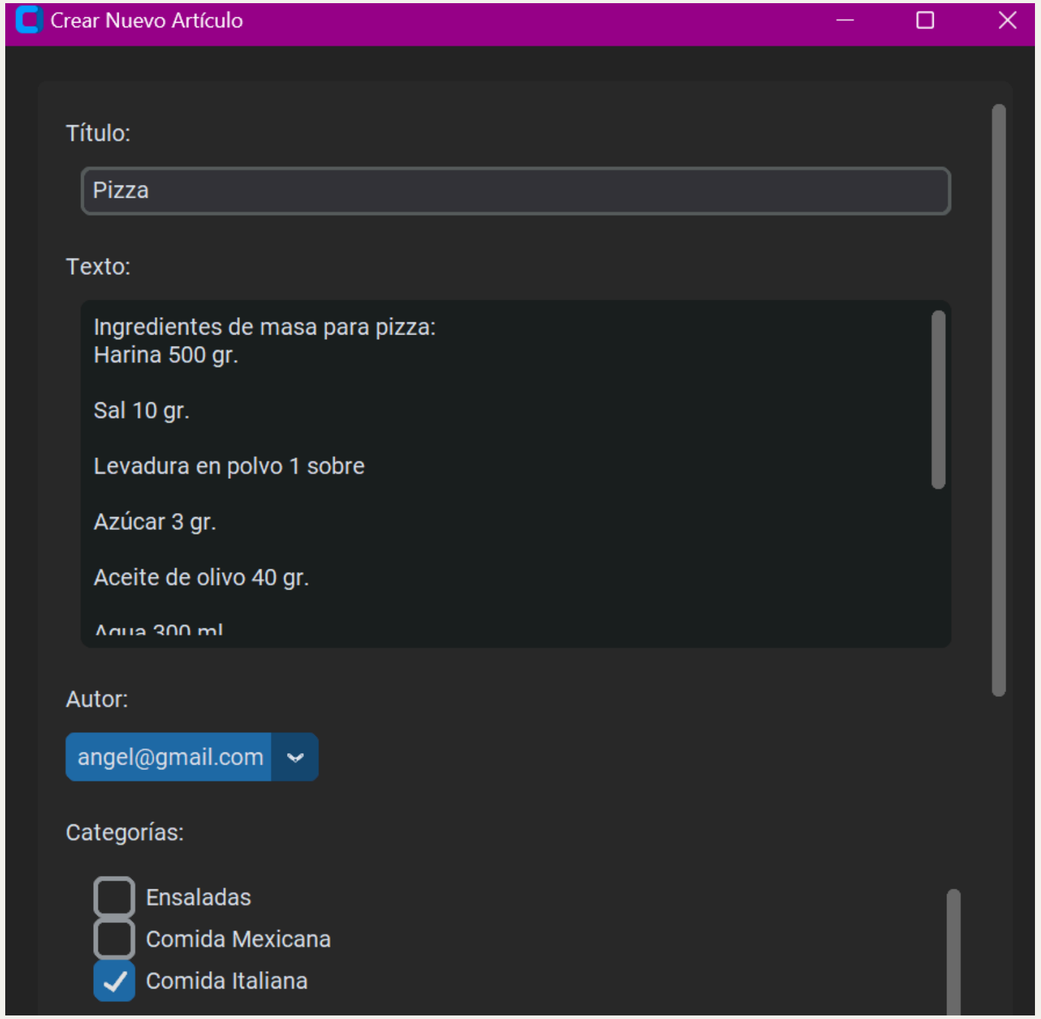
```python
def eliminar_uno(self, id_objeto):
    """Elimina un documento por su ObjectId."""
    try:
        resultado = self.coleccion.delete_one({"_id": id_objeto})
        self.cargar_mapa() # Recarga el mapa para eliminar la referencia del eleme
        return resultado.deleted_count # Retorna cuántos documentos fueron elimina
    except Exception as e:
        print(f"Error eliminando el documento en {self.coleccion.name}: {e}")
        return 0
```

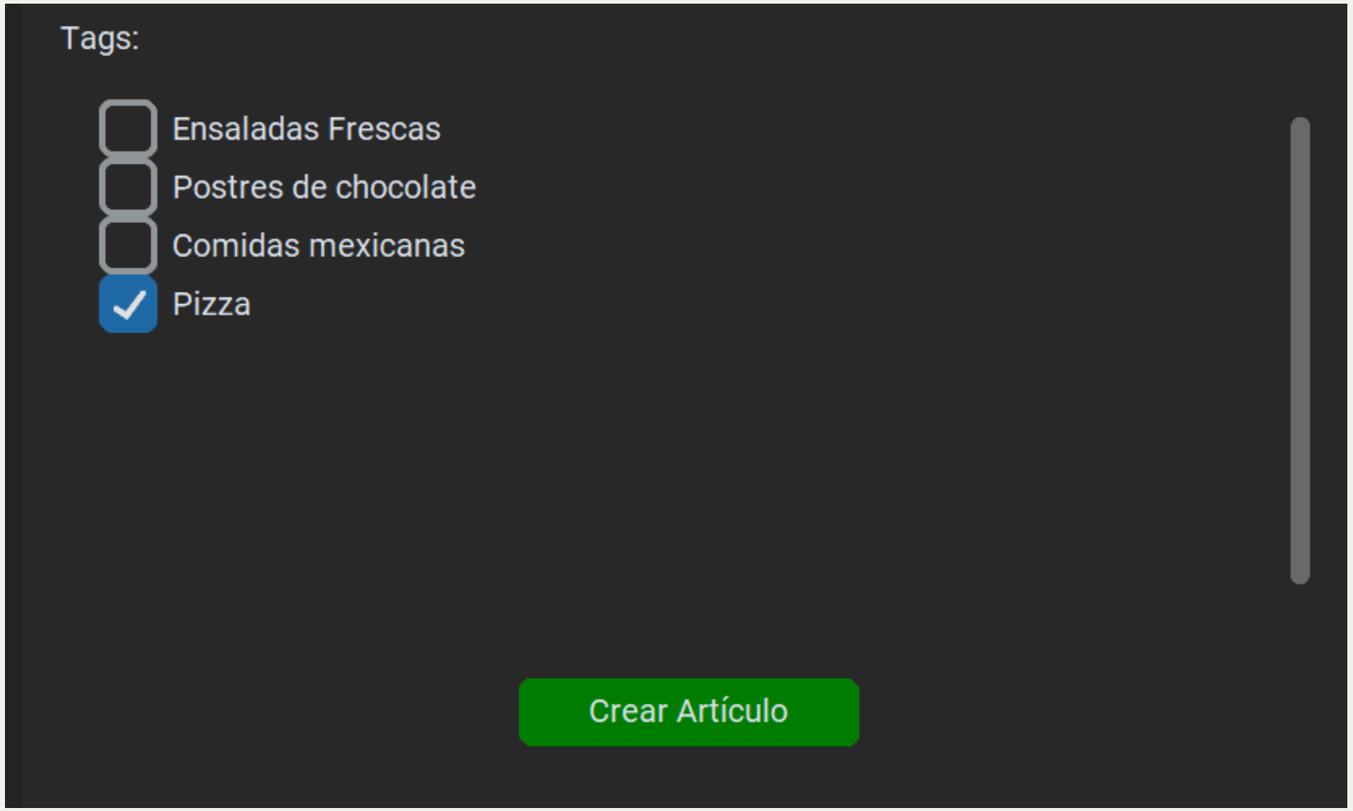
Llama a self.coleccion.delete_one({"_id": id_objeto}). También recarga el mapa para eliminar la referencia del elemento borrado.

RESULTADOS

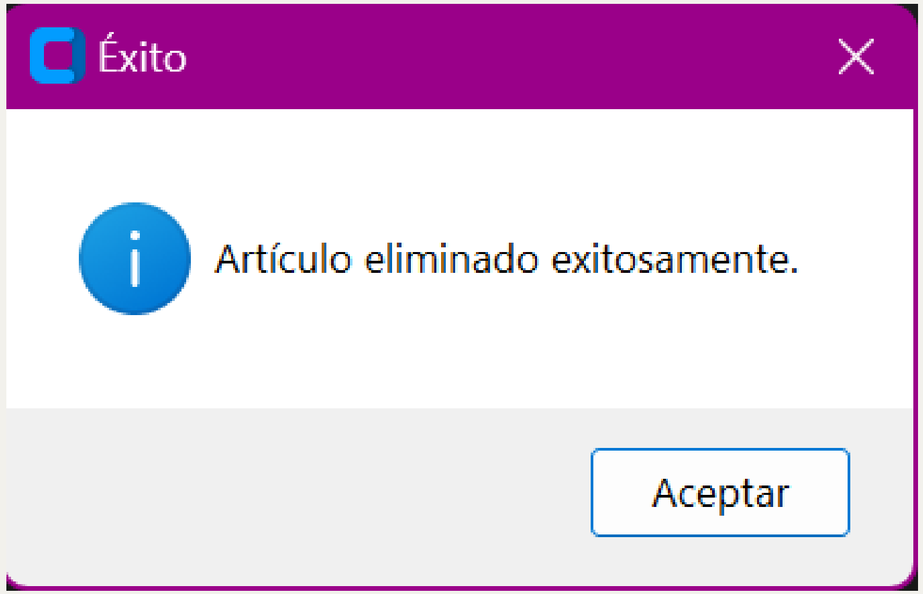

Gestión CRUD - Blog de Recetas
Menú Principal
1. Artículos (CRUD)
2. Categorías (CRUD)
3. Tags (CRUD)
4. Usuarios (CRUD)
Salir
Gestión de Artículos
Buscar por título, texto, autor, categoría o tag...
Buscar / Recargar
Crear Nuevo
--- Ensalada ---
ID: 690ec114fb0fc5d971561016
Fecha: 2025-11-07 22:03
Autor: angel@gmail.com
Categorías: Ensaladas
Tags: Ensaladas Frescas
Texto: Ingredientes:
Zanahoria
Lechuga
Mayonesa
Pollo
Intrucciones:
Ponemo...
--- Pastel ---
ID: 690ec741677e78e12a8b8809
Fecha: 2025-11-07 22:29
Autor: angel@gmail.com
Categorías: Ninguna
Tags: Postres de chocolate
Texto: Ingredientes
Porciones: 16
1 1/2 Barras de Mantequilla a temperatura ...
--- Enchiladas ---
ID de Artículo para acción
Editar Artículo
Eliminar Artículo

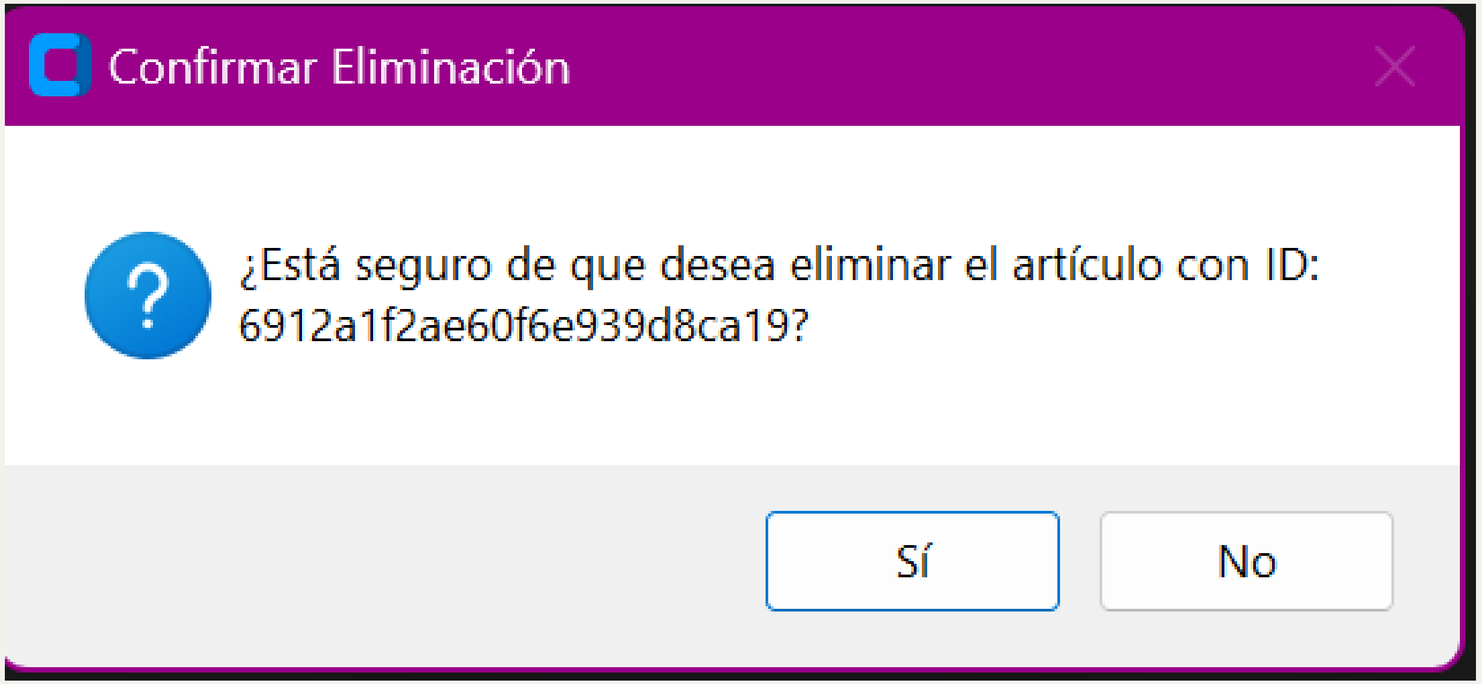
RESULTADOS


Crear Nuevo Artículo
Título:
Pizza
Texto:
Ingredientes de masa para pizza:
Harina 500 gr.
Sal 10 gr.
Levadura en polvo 1 sobre
Azúcar 3 gr.
Aceite de olivo 40 gr.
Agua 300 ml
Autor:
angel@gmail.com
Categorías:
Ensaladas
Comida Mexicana
Comida Italiana

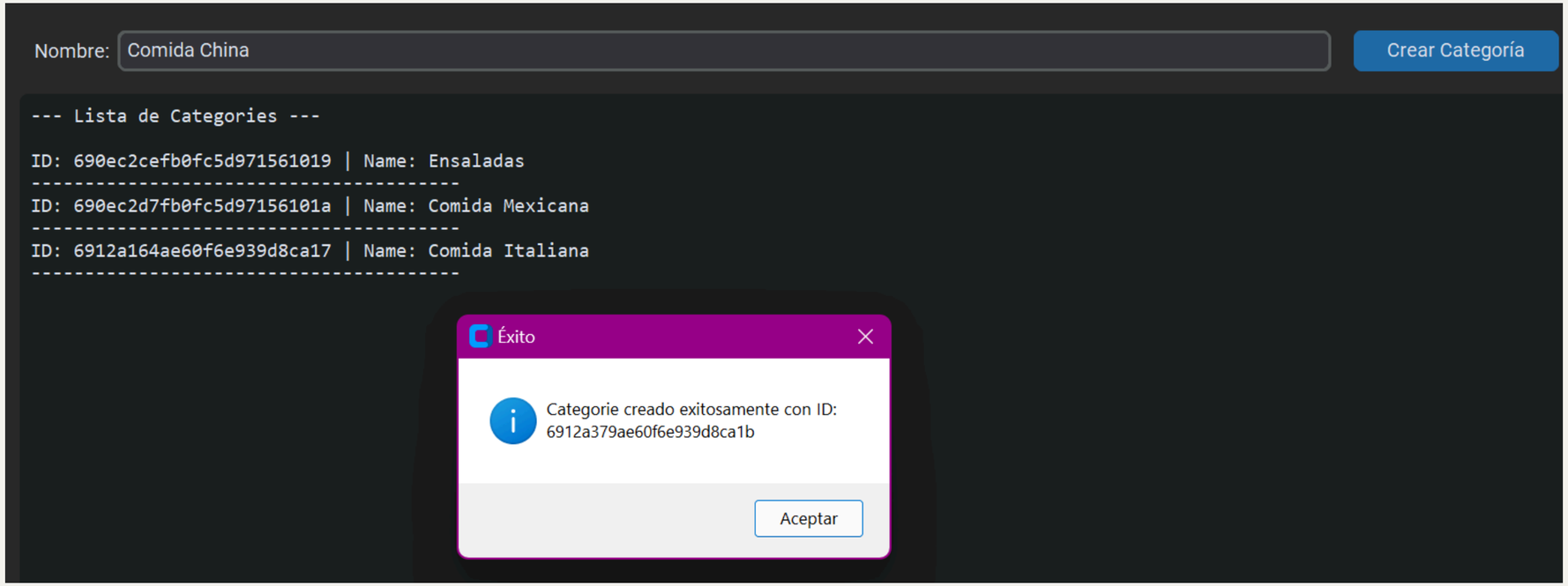

Tags:
Ensaladas Frescas
Postres de chocolate
Comidas mexicanas
Pizza
Crear Artículo

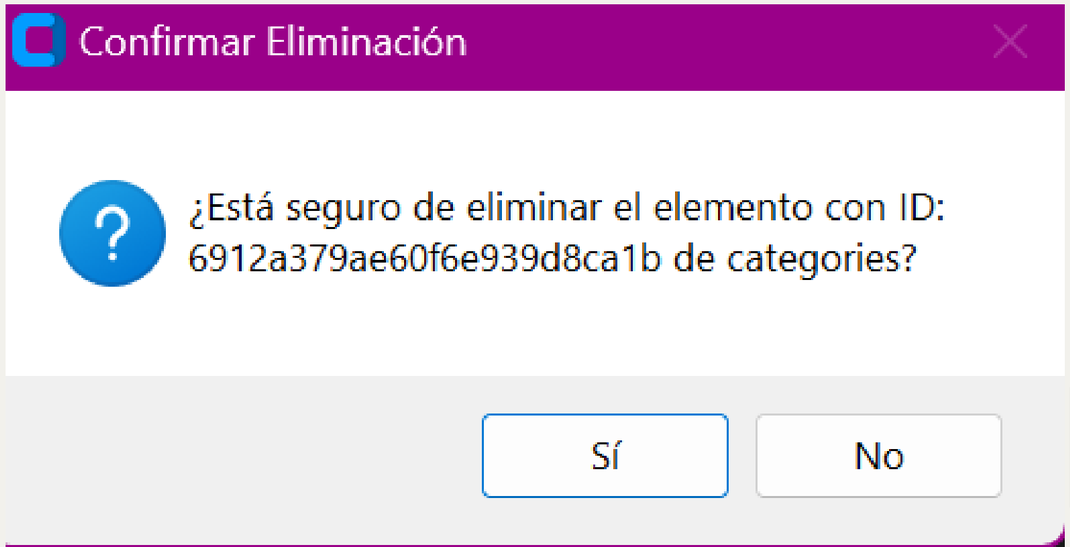
RESULTADOS

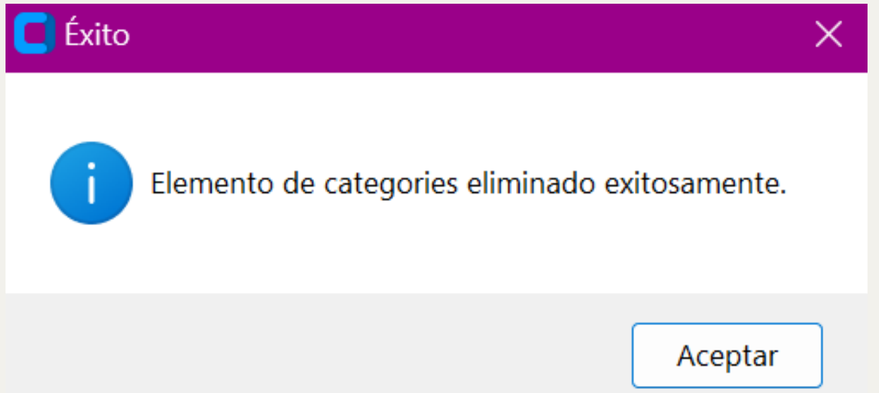

Éxito
Artículo eliminado exitosamente.
Aceptar


Confirmar Eliminación
¿Está seguro de que desea eliminar el artículo con ID: 6912a1f2ae60f6e939d8ca19?
Sí
No

RESULTADOS

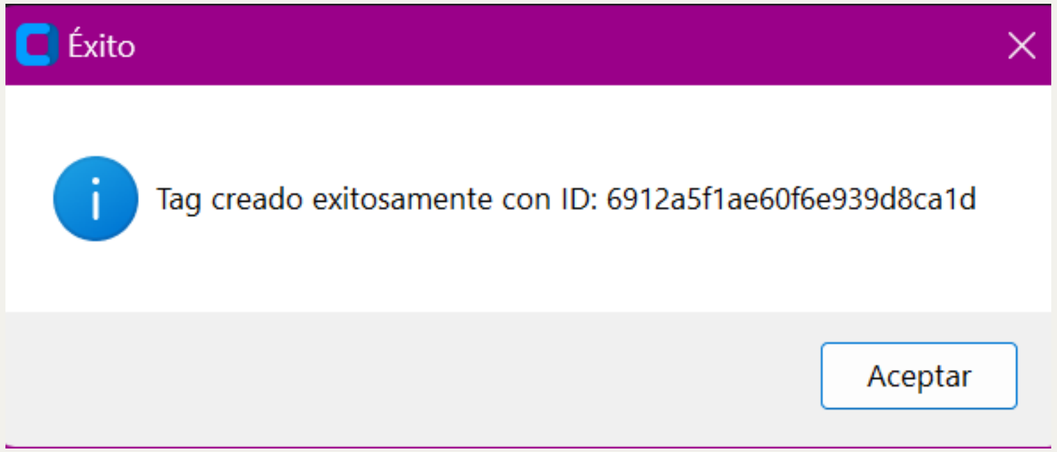

Nombre:
Comida China
Crear Categoría
--- Lista de Categories ---
ID: 690ec2cefb0fc5d971561019 | Name: Ensaladas
ID: 690ec2d7fb0fc5d97156101a | Name: Comida Mexicana
ID: 6912a164ae60f6e939d8ca17 | Name: Comida Italiana
Éxito
Categorie creado exitosamente con ID: 6912a379ae60f6e939d8ca1b
Aceptar

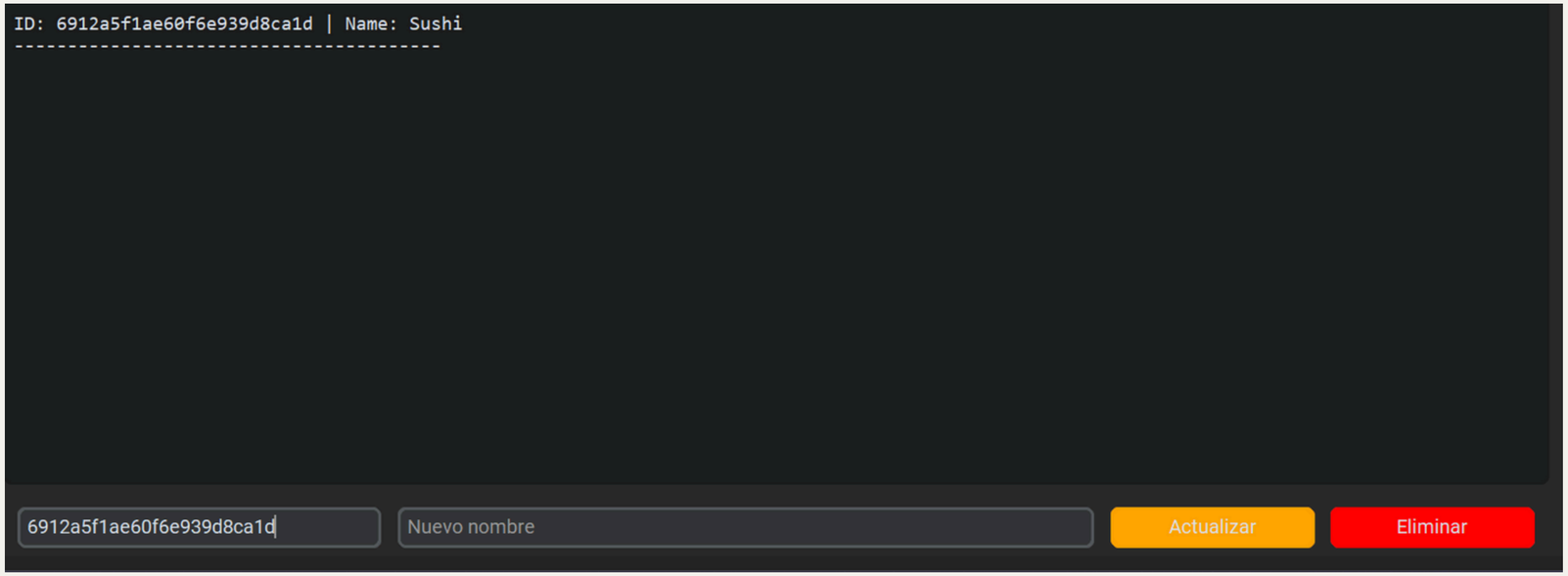

Confirmar Eliminación
¿Está seguro de eliminar el elemento con ID: 6912a379ae60f6e939d8ca1b de categories?
Sí
No


Éxito
Elemento de categories eliminado exitosamente.
Aceptar


690ec741677e78e12a8b8809
Editar Artículo
Eliminar Artículo

RESULTADOS


Éxito
Tag creado exitosamente con ID: 6912a5f1ae60f6e939d8ca1d
Aceptar


ID: 6912a5f1ae60f6e939d8ca1d | Name: Sushi
6912a5f1ae60f6e939d8ca1d
Nuevo nombre
Actualizar
Eliminar

INSERCIÓN Y SIMULACIÓN DE RELACIONES (JOINS)

Inserción de Artículo con Relaciones (menu.py)

```
16  class AppMenuPrincipal:
400     def guardar_articulo(self, ventana, id_articulo, titulo, texto, nombre_usuario, mapa_vars_categor
401         """Lógica para guardar (crear o editar) un artículo en la base de datos."""
402         if not titulo or not texto or nombre_usuario == "No hay usuarios":
403             messagebox.showwarning("Campos Requeridos", "El Título, Texto y Autor son obligatorios.",
404             return
405
406         try:
407             # Obtiene el ObjectId del autor usando el mapa cache del gestor.
408             id_usuario = gestor_usuarios.obtener_id_por_nombre(nombre_usuario)
409             if not id_usuario:
410                 messagebox.showerror("Error", "Autor no válido.", parent=ventana)
411                 return
412
413             # Filtra y recolecta los IDs de las categorías y tags marcados.
414             ids_categorias = [ _id for var, _id in mapa_vars_categoria if var.get() == 1 ]
415             ids_tags = [ _id for var, _id in mapa_vars_tag if var.get() == 1 ]
416
417             # Crea el documento base.
418             datos_nuevo_articulo = {
419                 "title": titulo,
420                 "text": texto,
421                 "user_id": id_usuario,
422                 "categories": ids_categorias,
423                 "tags": ids_tags
424             }
425
```

```
16  class AppMenuPrincipal:
400     def guardar_articulo(self, ventana, id_articulo, titulo, texto, nombre_usuario, mapa_vars_categor
426             if id_articulo:
427                 # Si hay ID, es una ACTUALIZACIÓN
428                 datos_nuevo_articulo["last_modified"] = datetime.datetime.now()
429                 self.coleccion_articulos.update_one({"_id": id_articulo}, {"$set": datos_nuevo_articul
430                 messagebox.showinfo("Éxito", "Artículo actualizado exitosamente.", parent=ventana)
431             else:
432                 # Si no hay ID, es una CREACIÓN
433                 datos_nuevo_articulo["date"] = datetime.datetime.now()
434                 self.coleccion_articulos.insert_one(datos_nuevo_articulo)
435                 messagebox.showinfo("Éxito", "Artículo creado exitosamente.", parent=ventana)
436
437             ventana.destroy() # Cierra la ventana modal.
438             self.cargar_articulos() # Recarga la lista principal para ver los cambios.
439         except Exception as e:
440             messagebox.showerror("Error de Guardado", f"No se pudo guardar el artículo:\n{e}", parent=
441
```

1. Primero, se obtienen los ObjectId de las categorías y tags marcados en los checkboxes .
2. Luego, se crea un diccionario datos_nuevo_articulo que incluye las referencias a otras colecciones: "user_id": id_usuario (1-a-N) y "categories": ids_categorias, "tags": ids_tags (N-a-N).
3. Finalmente, se inserta el documento con self.coleccion_articulos.insert_one(datos_nuevo_articulo).

SIMULACIÓN DE RELACIONES (JOINS)

Inserción de Artículo con Relaciones (menu.py)

```
16  class AppMenuPrincipal:
400     def guardar_articulo(self, ventana, id_articulo, titulo, texto, nombre_usuario, mapa_vars_categor
401         """Lógica para guardar (crear o editar) un artículo en la base de datos."""
402         if not titulo or not texto or nombre_usuario == "No hay usuarios":
403             messagebox.showwarning("Campos Requeridos", "El Título, Texto y Autor son obligatorios.",
404             return
405
406         try:
407             # Obtiene el ObjectId del autor usando el mapa cache del gestor.
408             id_usuario = gestor_usuarios.obtener_id_por_nombre(nombre_usuario)
409             if not id_usuario:
410                 messagebox.showerror("Error", "Autor no válido.", parent=ventana)
411                 return
412
413             # Filtra y recolecta los IDs de las categorías y tags marcados.
414             ids_categorias = [ _id for var, _id in mapa_vars_categoria if var.get() == 1 ]
415             ids_tags = [ _id for var, _id in mapa_vars_tag if var.get() == 1 ]
416
417             # Crea el documento base.
418             datos_nuevo_articulo = {
419                 "title": titulo,
420                 "text": texto,
421                 "user_id": id_usuario,
422                 "categories": ids_categorias,
423                 "tags": ids_tags
424             }
425
```

```
16  class AppMenuPrincipal:
400     def guardar_articulo(self, ventana, id_articulo, titulo, texto, nombre_usuario, mapa_vars_categor
426             if id_articulo:
427                 # Si hay ID, es una ACTUALIZACIÓN
428                 datos_nuevo_articulo["last_modified"] = datetime.datetime.now()
429                 self.coleccion_articulos.update_one({"_id": id_articulo}, {"$set": datos_nuevo_articul
430                 messagebox.showinfo("Éxito", "Artículo actualizado exitosamente.", parent=ventana)
431             else:
432                 # Si no hay ID, es una CREACIÓN
433                 datos_nuevo_articulo["date"] = datetime.datetime.now()
434                 self.coleccion_articulos.insert_one(datos_nuevo_articulo)
435                 messagebox.showinfo("Éxito", "Artículo creado exitosamente.", parent=ventana)
436
437             ventana.destroy() # Cierra la ventana modal.
438             self.cargar_articulos() # Recarga la lista principal para ver los cambios.
439         except Exception as e:
440             messagebox.showerror("Error de Guardado", f"No se pudo guardar el artículo:\n{e}", parent=
441
```

1. Primero, se obtienen los ObjectId de las categorías y tags marcados en los checkboxes .
2. Luego, se crea un diccionario datos_nuevo_articulo que incluye las referencias a otras colecciones: "user_id": id_usuario (1-a-N) y "categories": ids_categorias, "tags": ids_tags (N-a-N).
3. Finalmente, se inserta el documento con self.coleccion_articulos.insert_one(datos_nuevo_articulo).