



4.22 PROYECTO FINAL:

3RA. EVALUACIÓN.

**UNIVERSIDAD AUTONOMA DE CHIHUAHUA.
FACULTAD DE INGENIERIA.**

368027 *Sandoval Granados Paloma Ivonne.*

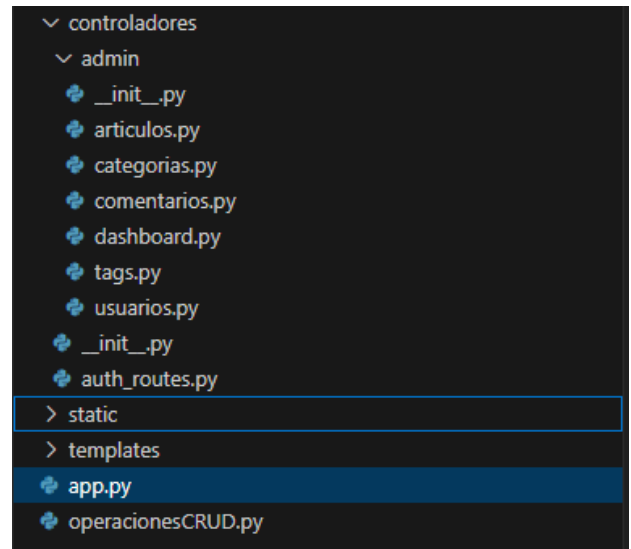
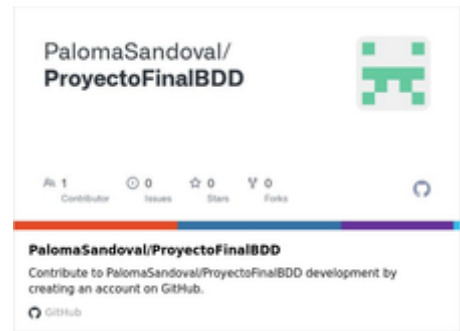
Docente: Jose Saul de Lira Miramontes.
BASES DE DATOS AVANZADAS - 7CC2.

23 Noviembre, 2025

Aplicación web de gestión para un blog construido en Python usando el micro-framework Flask.

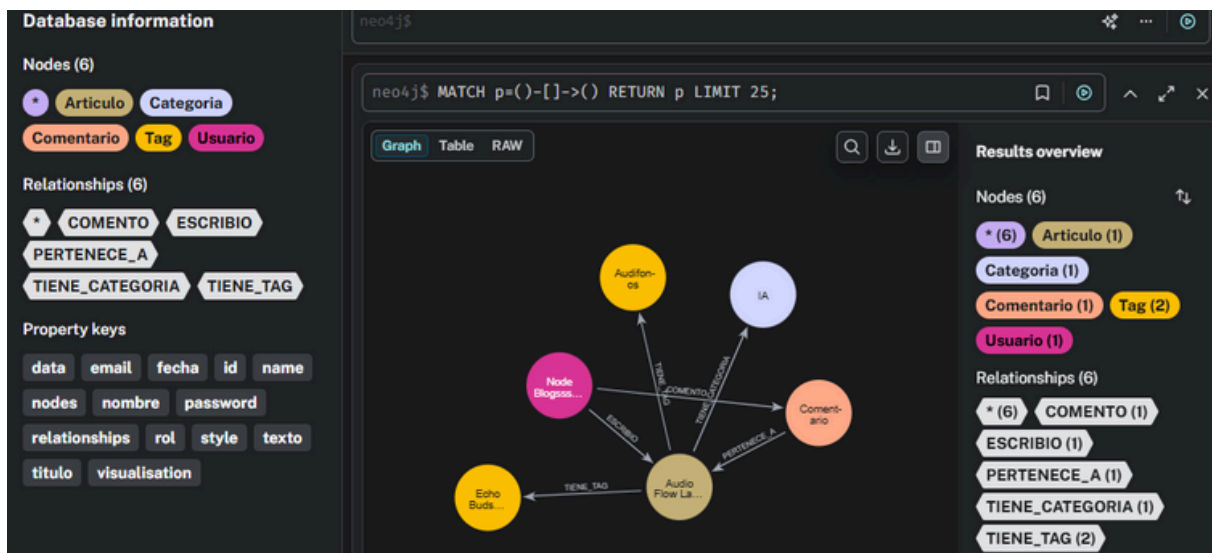
Arquitectura de la aplicación.

- operacionesCRUD.py:
Maneja la comunicación con la base de datos.
- controladores/:
Maneja las rutas (URLs) y la lógica de qué hacer cuando un usuario visita una página.
- templates/:
Contiene todos los archivos HTML que el usuario ve en el navegador.
- static/:
Almacena los archivos CSS (estilos) e imágenes.
- app.py:
Es el punto de entrada de la aplicación.



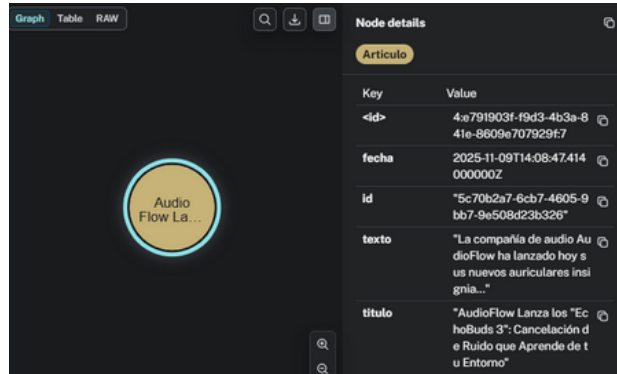
Base de datos.

La base de datos se encuentra almacenada en Neo4j Aura con el nombre de instancia “bdd” y el nombre de la base por default neo4j



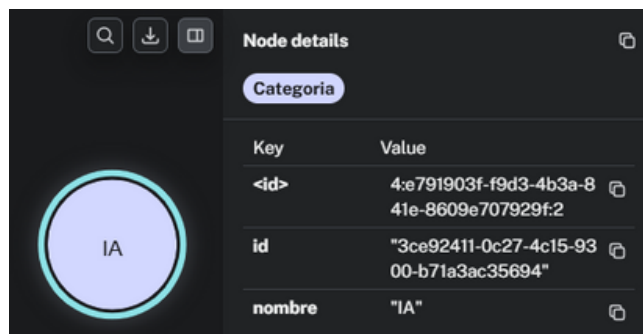
Cuenta con 5 nodos los cuales son artículos, categorías, comentarios, tags y usuarios, cada una dividida de la siguiente manera, con los ejemplos:

Artículos:



Key	Value
<id>	4:e791903f-f9d3-4b3a-841e-8609e707929f:7
fecha	2025-11-09T14:08:47.41400000Z
id	"5c70b2a7-6cb7-4605-9bb7-9e508d23b326"
texto	"La compañía de audio AudioFlow ha lanzado hoy sus nuevos auriculares insignia..."
titulo	"AudioFlow Lanza los 'EchoBuds 3': Cancelación de Ruido que Aprende de tu Entorno"

Categorías:



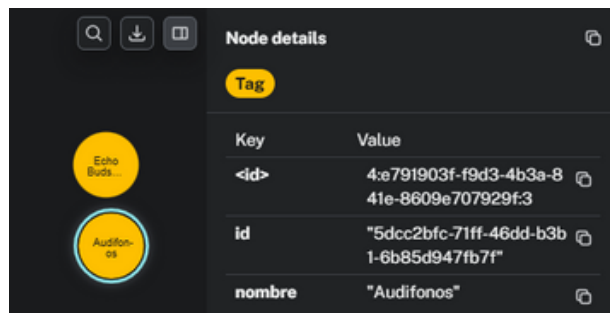
Key	Value
<id>	4:e791903f-f9d3-4b3a-841e-8609e707929f:2
id	"3ce92411-0c27-4c15-9300-b71a3ac35694"
nombre	"IA"

Comentarios:



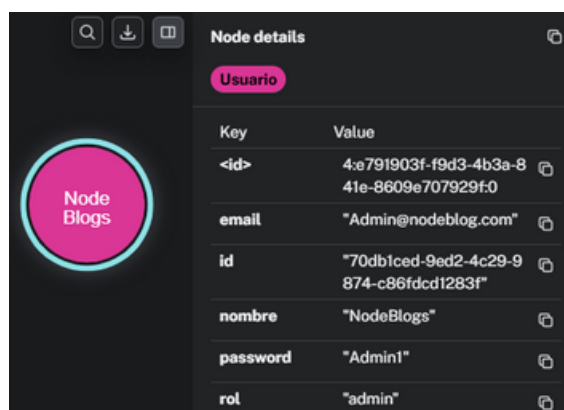
Key	Value
<id>	4:e791903f-f9d3-4b3a-841e-8609e707929f:8
fecha	2025-11-10T22:42:00.12300000Z
id	"695ea012-716a-42f7-ae f2-319eda905fca"
texto	"¡Qué buen artículo! Me sirvió mucho la explicación."

Tags:



Key	Value
<id>	4:e791903f-f9d3-4b3a-841e-8609e707929f:3
id	"5dcc2bfc-71ff-46dd-b3b1-6b85d947fb7f"
nombre	"Audifonos"

Usuarios:



Key	Value
<id>	4:e791903f-f9d3-4b3a-841e-8609e707929f:0
email	"Admin@nodeblog.com"
id	"70db1ced-9ed2-4c29-9874-c86fdcd1283f"
nombre	"NodeBlogs"
password	"Admin1"
rol	"admin"

Relaciones entre la base de datos.

Se cumplen las relaciones de dos maneras

Guardando el `_id` de un documento dentro de otro.

Usando el comando `$lookup` para juntar las colecciones al momento, simulando a un join en sql.

Uno a Muchos

Usuarios → Artículos (Uno-a-Muchos)

Un Usuario puede escribir muchos Artículos.

Pero un Artículo solo tiene un `user_id`.

Usuarios → Comentarios (Uno-a-Muchos)

Un Usuario tiene muchos Comentarios.

Pero un Comentario tiene un solo `user_id`.

Artículos → Comentarios (Uno-a-Muchos)

Un Artículo puede tener muchos Comentarios

Pero un Comentario solo pertenece a un artículo (tiene un solo `article_id`).

Muchos a Muchos

Artículos ⇔ Categorías (Muchos-a-Muchos)

Un Artículo puede estar en varias Categorías al mismo tiempo.

Y al mismo tiempo, una Categoría puede usar en artículos diferentes.

Artículos ⇔ Tags (Muchos-a-Muchos)

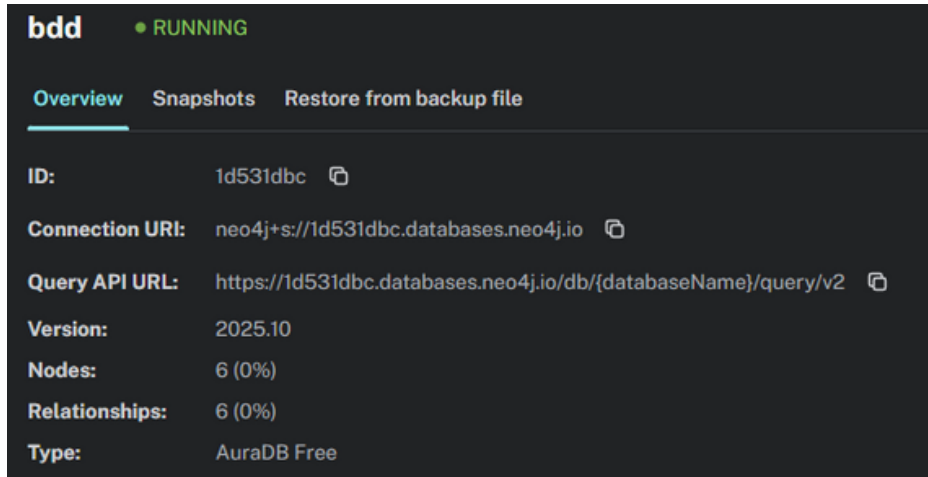
Un Artículo puede tener un muchos Tags

Y un Tag se le puede agregar a varios articulos.

Conexión a la base de datos.

La conexión desde Neo4j Aura se basa en el nombre de la instancia que nos ofrece Aura gratuito, el string de conexión creado es la siguiente:

La contraseña es la que ofrece aura al crear la base se crea la base de datos y la instancia, además del URI.



bdd	• RUNNING
Overview Snapshots Restore from backup file	
ID:	1d531dbc
Connection URI:	neo4j+s://1d531dbc.databases.neo4j.io
Query API URL:	https://1d531dbc.databases.neo4j.io/db/{databaseName}/query/v2
Version:	2025.10
Nodes:	6 (0%)
Relationships:	6 (0%)
Type:	AuraDB Free

La conexión desde python es de la siguiente manera:

Se define la URI y el usuario/contraseña

```
#Conexion
# URI de Aura
URI = "neo4j+s://1d531dbc.databases.neo4j.io"

# Usuario y contraseña
AUTH = ("neo4j", "CUX4IPic0UntZiNa_oij6zTd2_rMOAlZEBC2aX1Bk8A")

driver = GraphDatabase.driver(URI, auth=AUTH)
```

Funciones CRUD (OperacionesCRUD.py)

Se importan la librerías: from neo4j import GraphDatabase, from datetime import datetime e import uuid.

FUNCIONES DE CREACIÓN (CREATE)

1. registrar_usuario(nombre, email, password)

```
def registrar_usuario(nombre, email, password):
    #Checar si ya existe
    check_query = """
    MATCH (u:Usuario)
    WHERE u.email = $email OR u.nombre = $nombre
    RETURN u
    """

    existentes = ejecutar_query(check_query, {"email": email, "nombre": nombre})
    # Si devuelve algo, es que ya existe
    if existentes:
        return "El email o nombre de usuario ya está en uso."
    # Si no existe, se crea
    create_query = """
    CREATE (u:Usuario {
        id: randomUUID(),
        nombre: $nombre,
        email: $email,
        password: $password,
        rol: "admin"
    })
    RETURN u.id as id
    """

    result = ejecutar_query(create_query, {"nombre": nombre, "email": email, "password": password})
    # Si se creó, devuelve el ID
    if result:
        return result[0]['id']
    return None
```

2. agregar_comentario(article_id_str, user_id_str, texto_comentario)

```
def agregar_comentario(article_id, user_id, texto_comentario):
    query = """
    MATCH (u:Usuario {id: $user_id})
    MATCH (a:Articulo {id: $article_id})
    CREATE (c:Comentario {
        id: randomUUID(),
        texto: $texto,
        fecha: datetime()
    })
    CREATE (u)-[:COMENTO]->(c)
    CREATE (c)-[:PERTENECE_A]->(a)
    RETURN c.id as id
    """

    result = ejecutar_query(query, {"user_id": user_id, "article_id": article_id, "texto": texto_comentario})

    if result:
        return result[0]['id']
    return None
```

3. crear_categoria(nombre)

```
def crear_categoria(nombre):  
    query = """  
    MERGE (c:Categoria {nombre: $nombre})  
    ON CREATE SET c.id = randomUUID()  
    RETURN c.id as id  
    """  
  
    result = ejecutar_query(query, {"nombre": nombre})  
    return result[0]['id'] if result else None
```

4. crear_articulo(user_id, titulo, texto, ids_categorias, ids_tags)

```
def crear_articulo(user_id, titulo, texto, ids_categorias, ids_tags):  
    # Usuario -> Artículo -> Tags/Categorias  
    query = """  
    MATCH (u:Usuario {id: $user_id})  
    // Crear el artículo  
    CREATE (a:Artículo {  
        id: randomUUID(),  
        titulo: $titulo,  
        texto: $texto,  
        fecha: datetime()  
    })  
    CREATE (u)-[:ESCRIBIO]->(a)  
  
    // Conectar categorías  
    WITH a, $ids_categorias as cats, $ids_tags as tags  
    UNWIND cats as cat_id  
    MATCH (c:Categoria {id: cat_id})  
    CREATE (a)-[:TIENE_CATEGORIA]->(c)  
  
    // Conectar tags  
    WITH a, tags  
    UNWIND tags as tag_id  
    MATCH (t:Tag {id: tag_id})  
    CREATE (a)-[:TIENE_TAG]->(t)  
  
    RETURN a.id as id  
    """  
  
    result = ejecutar_query(query, {  
        "user_id": user_id,  
        "titulo": titulo,  
        "texto": texto,  
        "ids_categorias": ids_categorias,  
        "ids_tags": ids_tags  
    })  
    if result:  
        return result[0]['id']  
    return None
```

5. crear_tag(nombre)

```
def crear_tag(nombre):  
    query = """  
    MERGE (t:Tag {nombre: $nombre})  
    ON CREATE SET t.id = randomUUID()  
    RETURN t.id as id  
    """  
    result = ejecutar_query(query, {"nombre": nombre})  
    return result[0]['id'] if result else None
```


FUNCIONES DE LECTURA (READ)

1. iniciar_sesion(nombre, email, password)

```
def iniciar_sesion(email, password):
    print(f"--> Intentando Login: Email='{email}'")

    query = """
    MATCH (u:Usuario {email: $email, password: $password})
    RETURN u
    """

    result = ejecutar_query(query, {"email": email, "password": password})

    if result:
        print("--> Login Exitoso")
        return result[0]['u']
    else:
        print("--> Login Fallido: Email o contraseña incorrectos")
        return None
```

2.Articulos_blog()

```
def Articulos_blog():
    query = """
    MATCH (a:Articulo)-[:ESCRIBIO]-(u:Usuario)
    OPTIONAL MATCH (a)-[:TIENE_CATEGORIA]->(c:Categoria)
    OPTIONAL MATCH (a)-[:TIENE_TAG]->(t:Tag)
    RETURN
        a.id as _id,
        a.titulo as titulo,
        toString(a.fecha) as fecha,
        a.texto as texto,
        u.nombre as autor_nombre,
        collect(DISTINCT c.nombre) as categorias,
        collect(DISTINCT t.nombre) as tags
    ORDER BY fecha DESC
    """

    return ejecutar_query(query)
```

3. obtener_todos_comentarios()

```
def obtener_todos_comentarios():
    query = """
    MATCH (c:Comentario)-[:PERTENECE_A]->(a:Articulo)
    MATCH (u:Usuario)-[:COMENTO]->(c)
    RETURN
        c.id as _id,
        c.texto as texto_comentario,
        toString(c.fecha) as fecha,
        u.nombre as autor_nombre,
        a.titulo as articulo_titulo
    ORDER BY fecha DESC
    """

    return ejecutar_query(query)
```

4. obtener_todas_categorias() / obtener_todos_tags()

```
def obtener_todas_categorias():  
    return ejecutar_query("MATCH (c:Categoria) " \  
        "RETURN c.id as _id, c.nombre as name")
```

```
def obtener_todos_tags():  
    return ejecutar_query("MATCH (t:Tag) RETURN " \  
        "t.id as _id, t.nombre as name")
```

5. obtener_todos_usuarios

```
def obtener_todos_usuarios():  
    return ejecutar_query("MATCH (u:Usuario) RETURN " \  
        "u.id as _id, u.nombre as name, u.email as email," \  
        " u.rol as role ORDER BY u.nombre")
```

6. obtener_articulo_por_id

```
def obtener_articulo_por_id(article_id):  
    query = """  
    MATCH (a:Articulo {id: $id})  
    OPTIONAL MATCH (a)-[:TIENE_CATEGORIA]->(c:Categoria)  
    OPTIONAL MATCH (a)-[:TIENE_TAG]->(t:Tag)  
    RETURN  
        a.id as _id,  
        a.titulo as title,  
        a.texto as text,  
        collect(c.id) as categories,  
        collect(t.id) as tags,  
        a.autor_nombre as autor_nombre  
    """  
    result = ejecutar_query(query, {"id": article_id})  
    return result[0] if result else None
```

7. obtener_mi_perfil

```
def obtener_mi_perfil(user_id):  
    query = """  
    MATCH (u:Usuario {id: $id})  
    RETURN u.id as _id, u.nombre as name, u.email as email,  
    u.password as password  
    """  
    result = ejecutar_query(query, {"id": user_id})  
    return result[0] if result else None
```

8. obtener_categoria_por_id

```
def obtener_categoria_por_id(cat_id):  
    query = """  
    MATCH (t:Categoria {id: $id})  
    RETURN D  
        t.id as _id,  
        t.nombre as name  
    """  
    result = ejecutar_query(query, {"id": cat_id})  
    return result[0] if result else None
```

9. obtener_tag_por_id

```
def obtener_tag_por_id(tag_id):  
    query = """  
    MATCH (t:Tag {id: $id})  
    RETURN  
        t.id as _id,          // El HTML pide ._id  
        t.nombre as name     // El HTML pide .name  
    """
```

10. obtener_comentario_por_id

```
def obtener_comentario_por_id(com_id):  
    query = """  
    MATCH (c:Comentario {id: $id})  
    RETURN  
        c.id as _id,  
        c.texto as text,  
        toString(c.fecha) as fecha  
    """  
  
    result = ejecutar_query(query, {"id": com_id})  
    return result[0] if result else None
```

FUNCIONES DE ACTUALIZACION (UPDATE)

1. actualizar_perfil

```
def actualizar_perfil(user_id, nombre, email, password):
    # Si la password no tiene ndaa, no se cambia para no borrar
    if password:
        query = """
        MATCH (u:Usuario {id: $id})
        SET u.nombre = $nombre, u.email = $email, u.password = $password
        RETURN u.id
        """
        params = {"id": user_id, "nombre": nombre, "email": email, "password": password}
    else:
        # Si no escribió password nueva, solo actualiza nombre y email
        query = """
        MATCH (u:Usuario {id: $id})
        SET u.nombre = $nombre, u.email = $email
        RETURN u.id
        """
        params = {"id": user_id, "nombre": nombre, "email": email}

    result = ejecutar_query(query, params)
    return True if result else False
```

2. editar_articulo

```
def editar_articulo(article_id, titulo, texto, ids_categorias, ids_tags):
    query = """
    MATCH (a:Articulo {id: $id})
    SET a.titulo = $titulo, a.texto = $texto

    // Borrar relaciones viejas
    WITH a
    OPTIONAL MATCH (a)-[r1:TIENE_CATEGORIA]->()
    OPTIONAL MATCH (a)-[r2:TIENE_TAG]->()
    DELETE r1, r2

    // Crear nuevas relaciones
    WITH a
    UNWIND $ids_cats as cat_id
    MATCH (c:Categoria {id: cat_id})
    MERGE (a)-[:TIENE_CATEGORIA]->(c)

    WITH a
    UNWIND $ids_tags as tag_id
    MATCH (t:Tag {id: tag_id})
    MERGE (a)-[:TIENE_TAG]->(t)

    RETURN a.id
    """
    res = ejecutar_query(query, {
        "id": article_id,
        "titulo": titulo,
        "texto": texto,
        "ids_cats": ids_categorias,
        "ids_tags": ids_tags
    })
    return True if res else False
```

3. editar_comentario

```
def editar_comentario(comment_id, nuevo_texto):  
    query = "MATCH (c:Comentario {id: $id}) SET c.texto = $texto RETURN c.id"  
    res = ejecutar_query(query, {"id": comment_id, "texto": nuevo_texto})  
    return True if res else False
```

4. editar_categoria

```
def editar_categoria(cat_id, nuevo_nombre):  
    check = ejecutar_query("MATCH (c:Categoria {nombre: $nombre}) WHERE c.id <> $id RETURN c",  
                           {"nombre": nuevo_nombre, "id": cat_id})  
    if check: return False  
    query = "MATCH (c:Categoria {id: $id}) SET c.nombre = $nombre RETURN c.id"  
    res = ejecutar_query(query, {"id": cat_id, "nombre": nuevo_nombre})  
    return True if res else False
```

5. editar_tag

```
def editar_tag(tag_id, nuevo_nombre):  
    check = ejecutar_query("MATCH (t:Tag {nombre: $nombre}) WHERE t.id <> $id RETURN t",  
                           {"nombre": nuevo_nombre, "id": tag_id})  
    if check: return False  
  
    query = "MATCH (t:Tag {id: $id}) SET t.nombre = $nombre RETURN t.id"  
    res = ejecutar_query(query, {"id": tag_id, "nombre": nuevo_nombre})  
    return True if res else False
```

FUNCIONES DE ELIMINACION (DELETE)

1. eliminar_usuario

```
def eliminar_usuario(user_id):
    query = """
    MATCH (u:Usuario {id: $id})

    // Borrar comentarios hechos por él
    OPTIONAL MATCH (u)-[:COMENTO]->(c_propio:Comentario)
    DETACH DELETE c_propio

    // Borrar artículos y los comentarios que tengan esos artículos
    WITH u
    OPTIONAL MATCH (u)-[:ESCRIBIO]->(a:Articulo)
    OPTIONAL MATCH (a)-[:PERTENECE_A]-(c_ajeno:Comentario)
    DETACH DELETE c_ajeno, a

    // Finalmente borrar al usuario
    WITH u
    DETACH DELETE u
    RETURN count(u) as deleted
    """
    ejecutar_query(query, {"id": user_id})
    return True
```

2. eliminar_articulo

```
def eliminar_articulo(article_id):
    query = """
    MATCH (a:Articulo {id: $id})
    OPTIONAL MATCH (a)-[:PERTENECE_A]-(c:Comentario)
    DETACH DELETE c, a
    """
    ejecutar_query(query, {"id": article_id})
    return True
```

3. eliminar_comentario_individual

```
def eliminar_comentario_individual(comment_id):
    query = "MATCH (c:Comentario {id: $id}) DETACH DELETE c"
    ejecutar_query(query, {"id": comment_id})
    return True
```

4. eliminar_categoria

```
def eliminar_categoria(cat_id):
    query = "MATCH (c:Categoria {id: $id}) DETACH DELETE c"
    ejecutar_query(query, {"id": cat_id})
    return True
```

5. eliminar_tag

```
def eliminar_tag(tag_id):
    query = "MATCH (t:Tag {id: $id}) DETACH DELETE t"
    ejecutar_query(query, {"id": tag_id})
    return True
```

Framework web: Flujo de peticiones usando Flask.

El sistema opera bajo un patrón de diseño Modelo-Vista-Controlador.

Cada acción del usuario, ya sea agregar, editar o eliminar, sigue un ciclo de peticiones entre estos tres componentes.

Modelo con la BDD(CRUD):

Es el archivo **operacionesCRUD.py**. Es el único componente responsable de la lógica de con la base de datos MongoDB.

Vista (Interfaz de Usuario):

Son los archivos .html en la carpeta **templates/**. Su única función es mostrar datos al usuario (p.ej., {{ categoria.name }}) y capturar nuevas entradas (p.ej., <form>).

Controlador (Lógica de Aplicación): Son los archivos Python en la carpeta **controladores/** (p.ej., categorias.py, articulos.py). Actúan como el intermediario: reciben peticiones del usuario, solicitan datos al Modelo y deciden qué Vista mostrar.

Flujo de Solicitud de Datos (Método GET)

Este es el proceso para mostrar una página:

1. Petición de Usuario: El usuario hace clic en un enlace ("Editar"). El navegador envía una petición HTTP GET a una URL definida (p.ej., /admin/categoria/editar/<id>).
2. Enrutamiento (Flask): Flask intercepta la URL y la dirige a la función correspondiente en el Controlador (p.ej., admin_editar_categoria_ruta en categorias.py).
3. Consulta al Modelo: El Controlador llama a la función de lectura necesaria del Modelo CRUD (p.ej., obtener_categoria_por_id(id)) para buscar los datos en la base de datos.
4. Respuesta del Modelo: El Modelo devuelve los datos solicitados (el documento de la categoría) al Controlador.
5. Renderizado de Vista: El Controlador toma esos datos y los "renderiza" o inyecta en una plantilla de la Vista (p.ej., admin_editar_categoria.html). El HTML resultante se envía al navegador, y el usuario ve el formulario con los datos cargados.

Flujo de Procesamiento de Datos (Método POST)

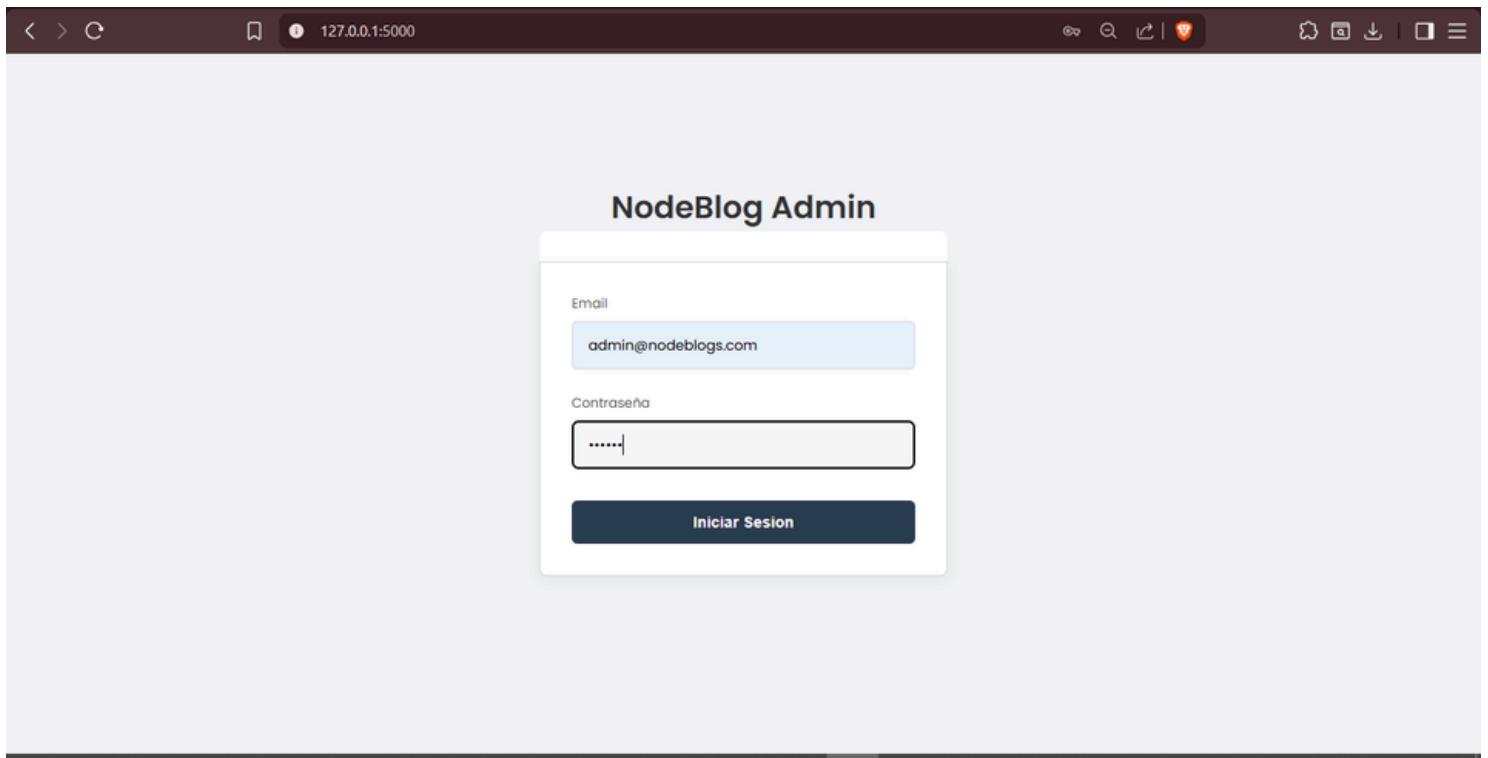
Este es el proceso para procesar una acción (guardar un cambio, crear un nuevo elemento o eliminar).

1. Envío de Formulario: El usuario presiona un botón (p.ej., "Confirmar Cambios"). El navegador empaqueta los datos del formulario y envía una petición HTTP POST a la URL definida en el <form>.
2. Enrutamiento (Flask): Flask dirige esta petición POST a la función correspondiente en el Controlador (a menudo la misma función que el GET, pero esta vez detecta `request.method == "POST"`).
3. Extracción de Datos: El Controlador extrae los datos enviados del formulario (p.ej., `request.form.get('nombre')`).
4. Acción en Modelo: El Controlador llama a la función de escritura del Modelo (p.ej., `editar_categoria(id, nuevo_nombre)`), pasando los nuevos datos.
5. Respuesta del Modelo: El Modelo ejecuta la operación en MongoDB (p.ej., `update_one`) y devuelve un estado de éxito (p.ej., `True`) al Controlador.
6. Redirección: Al recibir el éxito, el Controlador no renderiza una vista. En su lugar, emite una redirección (`redirect`). Esto le ordena al navegador que debe solicitar una nueva página (generalmente, la lista principal, p.ej., `/admin/categorias`).
7. Ciclo Completo: Esta redirección inicia un nuevo flujo GET (paso 1 del proceso anterior), lo que garantiza que el usuario vea la lista actualizada con los cambios recién guardados.

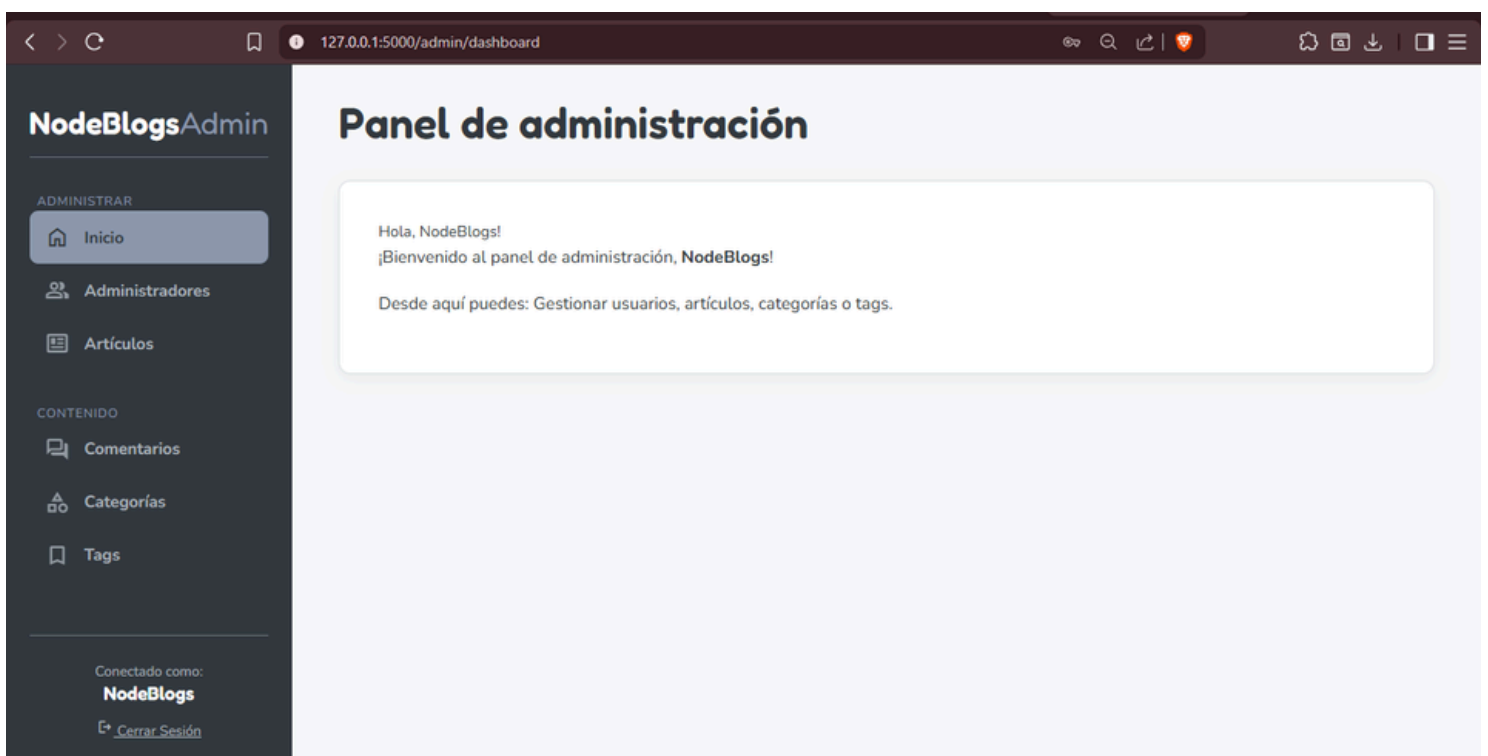
Correr servicio y app

```
C:\Users\palom\Documents\BasesVistaAdmin2 Ajuste - copia>app.py python
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 480-840-237
```


Flujo de la pagina web.



The screenshot shows the login page of the NodeBlog Admin interface. The browser's address bar displays the URL `127.0.0.1:5000`. The page features a central login form with the title "NodeBlog Admin". The form includes an "Email" field with the text `admin@nodeblogs.com`, a "Contraseña" (Password) field with masked characters, and a dark blue "Iniciar Sesión" (Login) button.



The screenshot shows the dashboard of the NodeBlog Admin interface. The browser's address bar displays the URL `127.0.0.1:5000/admin/dashboard`. The page has a dark sidebar on the left with the title "NodeBlogsAdmin" and a main content area on the right titled "Panel de administración".

ADMINISTRAR

- [Inicio](#)
- [Administradores](#)
- [Artículos](#)

CONTENIDO

- [Comentarios](#)
- [Categorías](#)
- [Tags](#)

Conectado como: **NodeBlogs**

[Cerrar Sesión](#)

Panel de administración

Hola, NodeBlogs!
¡Bienvenido al panel de administración, NodeBlogs!

Desde aquí puedes: Gestionar usuarios, artículos, categorías o tags.

NodeBlogsAdmin

ADMINISTRAR

[Inicio](#)

[Administradores](#)

[Artículos](#)

CONTENIDO

[Comentarios](#)

[Categorías](#)

[Tags](#)

Conectado como:
NodeBlogs

[Cerrar Sesión](#)

Gestión de Administradores

Dar de alta nuevo Admin

Nombre

Email

Contraseña

[Agregar a Nuevo Administrador](#)

Equipo de Administración

Nombre	Email	Acciones
NodeBlogs	Admin@nodeblog.com	Editar Eliminar

NodeBlogsAdmin

ADMINISTRAR

[Inicio](#)

[Administradores](#)

[Artículos](#)

CONTENIDO

[Comentarios](#)

[Categorías](#)

[Tags](#)

Conectado como:
NodeBlogs

[Cerrar Sesión](#)

Editando mi Perfil

Mis Datos Personales

Nombre

NodeBlogs

Correo Electrónico

Admin@nodeblog.com

Nueva Contraseña

Si no quieres cambiar la contraseña deja este campo vacío

[Guardar Mis Cambios](#)

Windows taskbar with icons for File Explorer, Edge, and other applications. System clock shows 04:07 p.m. on 23/11/2025.

NodeBlogsAdmin

ADMINISTRAR

- Inicio
- Administradores
- Artículos

CONTEIDO

- Comentarios
- Categorías
- Tags

Conectado como: **NodeBlogs**

[Cerrar Sesión](#)

Gestión de Artículos

[Crear Nuevo Artículo](#)

Título del Artículo	Autor	Categoría	Tags	Acciones
AudioFlow Lanza los "EchoBuds 3": Cancelación de Ruido que Aprende de tu Entorno	NodeBlogs	IA	#Audifonos #EchoBuds 3	Editar Eliminar

NodeBlogsAdmin

ADMINISTRAR

- Inicio
- Administradores
- Artículos

CONTEIDO

- Comentarios
- Categorías
- Tags

Conectado como: **NodeBlogs**

[Cerrar Sesión](#)

Editando Artículo

[Cancelar](#)

Título del Artículo

AudioFlow Lanza los "EchoBuds 3": Cancelación de Ruido que Aprende de tu Entorno

Contenido del Artículo

La compañía de audio AudioFlow ha lanzado hoy sus nuevos auriculares insignia...

Categorías

☒ IA

Tags

☒ #Audifonos ☒ #EchoBuds 3

NodeBlogsAdmin

ADMINISTRAR

- Inicio
- Administradores
- Artículos

CONTEIDO

- Comentarios
- Categorías
- Tags

Conectado como: **NodeBlogs**

[Cerrar Sesión](#)

Crear Nuevo Artículo

[Cancelar](#)

Título del Artículo

Contenido del Artículo

Categorías

☐ IA

Tags

☐ #Audifonos ☐ #EchoBuds 3

[Publicar Artículo](#)

127.0.0.1:5000/admin/comentarios

NodeBlogsAdmin

ADMINISTRAR

- Inicio
- Administradores
- Artículos

CONTENIDO

- Comentarios
- Categorías
- Tags

Conectado como: **NodeBlogs**
[Cerrar Sesión](#)

Gestión de Comentarios

Crear Nuevo Comentario

Asignar al Artículo: Texto del Comentario:

[Crear](#)

Comentario	Autor	En el Artículo	Fecha	Acción
¡Qué buen artículo! Me sirvió mucho la explicación.	NodeBlogs	AudioFlow Lanza los "EchoBuds 3": Cancelación de Ruido que Aprende de tu Entorno	2025-11-10 22:42	Editar Eliminar

127.0.0.1:5000/admin/comentario/editar/695ea012-716a-42f7-ae2-319eda905fca

NodeBlogsAdmin

ADMINISTRAR

- Inicio
- Administradores
- Artículos

CONTENIDO

- Comentarios
- Categorías
- Tags

Conectado como: **NodeBlogs**
[Cerrar Sesión](#)

Editando Comentario

[Cancelar](#)

Texto del Comentario

¡Qué buen artículo! Me sirvió mucho la explicación.

[Confirmar Cambios](#)

NodeBlogsAdmin

ADMINISTRAR

- Inicio
- Administradores
- Artículos

CONTENIDO

- Comentarios
- Categorías**
- Tags

Conectado como: **NodeBlogs**

[Cerrar Sesión](#)

Gestión de Categorías

Crear Nueva Categoría

Nombre de la Categoría

[Crear](#)

Categorías Existentes

Nombre de la Categoría	Acciones
IA	Editar Eliminar

Google Gemini

Usuarios Admin

Tarea

Proyectos - Canva

Reporte BDD 3ERA EVALUACIÓN

Editar Categoría

127.0.0.1:5000/admin/categoria/editar/3ce92411-0c27-4c15-9300-b71a3ac35694

NodeBlogsAdmin

ADMINISTRAR

- Inicio
- Administradores
- Artículos

CONTENIDO

- Comentarios
- Categorías**
- Tags

Conectado como: **NodeBlogs**

[Cerrar Sesión](#)

Editando Categoría

[Cancelar](#)

Nombre de la Categoría

[Confirmar Cambios](#)

04:09 p. m. 23/11/2025

Google Gemini Usuarios Admin Tarea Proyectos - Canva Reporte BDD 3ERA EVALU Tags

127.0.0.1:5000/admin/tags

NodeBlogsAdmin

ADMINISTRAR

- Inicio
- Administradores
- Artículos

CONTENIDO

- Comentarios
- Categorías
- Tags**

Conectado como: **NodeBlogs**
[Cerrar Sesión](#)

Gestión de Tags

Crear Nuevo Tag

Nombre del Tag

Crear

Tags Existentes

Nombre del Tag	Acciones
#Audifonos	Editar Eliminar
#EchoBuds 3	Editar Eliminar

04:10 p. m. 23/11/2025

Google Gemini Usuarios Admin Tarea Proyectos - Canva Reporte BDD 3ERA EVALU Editar Tag

127.0.0.1:5000/admin/tag/editar/5dcc2bfc-71ff-46dd-b3b1-6b85d947fb7f

NodeBlogsAdmin

ADMINISTRAR

- Inicio
- Administradores
- Artículos

CONTENIDO

- Comentarios
- Categorías
- Tags**

Conectado como: **NodeBlogs**
[Cerrar Sesión](#)

Editando Tag

Cancelar

Nombre del Tag (sin el #)

Audifonos

Confirmar Cambios

04:10 p. m. 23/11/2025