

Introducción a NoSQL



Que es una Base de datos Nosql



Una base de datos NoSQL es un tipo de sistema de almacenamiento de datos que no sigue el modelo relacional tradicional basado en tablas y esquemas rígidos, como es el caso de las bases de datos SQL (Structured Query Language). En su lugar, las bases de datos NoSQL están diseñadas para manejar grandes volúmenes de datos, estructuras de datos flexibles y aplicaciones distribuidas, ofreciendo mayor escalabilidad y flexibilidad.

Características principales de las bases de datos NoSQL:

1. **Modelo de Datos Flexibles:** Las bases de datos NoSQL no requieren esquemas predefinidos, lo que permite almacenar datos de forma no estructurada o semi-estructurada. Los datos pueden variar en estructura y formato, lo que facilita la adaptación a cambios en los requisitos de la aplicación.
2. **Alta Escalabilidad:** Son diseñadas para escalar horizontalmente, es decir, añadir más servidores para manejar incrementos en la carga de trabajo, en lugar de mejorar la capacidad de un solo servidor (escalado vertical).
3. **Tipos de Almacenamiento:**
 - **Bases de Datos de Clave-Valor:** Almacenan datos como pares clave-valor, donde la clave es un identificador único y el valor es el dato asociado (ej. Redis, DynamoDB).
 - **Bases de Datos de Documentos:** Almacenan datos en formatos de documentos, como JSON o BSON. Cada documento puede tener una estructura diferente (ej. MongoDB, CouchDB).
 - **Bases de Datos de Grafos:** Modelan datos como grafos con nodos y aristas, lo que es útil para representar relaciones complejas entre datos (ej. Neo4j).
 - **Bases de Datos de Columnas Anchas:** Organizan los datos en tablas con filas y columnas, pero a diferencia de las bases de datos relacionales, las columnas se pueden agrupar en familias y ser almacenadas de manera eficiente (ej. Cassandra, HBase).
4. **Tolerancia a Particiones:** Muchas bases de datos NoSQL están diseñadas para ser distribuidas en múltiples servidores o clusters, lo que permite manejar fallos en alguna parte del sistema sin que se interrumpa el servicio.
5. **Rendimiento Optimizado:** Están optimizadas para lecturas y escrituras rápidas, lo que las hace adecuadas para aplicaciones que requieren un alto rendimiento, como sistemas de recomendaciones, análisis de grandes volúmenes de datos en tiempo real, etc.

Las bases de datos NoSQL son populares en aplicaciones de big data, análisis en tiempo real, redes sociales, e-commerce, y cualquier otra aplicación que requiera manejar grandes volúmenes de datos de manera rápida y eficiente.



Cuando se habla de API, ¿a qué se refiere un endpoint?



Un **endpoint** en el contexto de una API (Interfaz de Programación de Aplicaciones) se refiere a una URL específica en un servidor que representa un recurso o una acción particular. Es el punto de acceso donde un cliente puede enviar solicitudes (requests) y recibir respuestas (responses) desde la API.

Desglose del concepto:

1. **API:** Es un conjunto de reglas y definiciones que permite a diferentes aplicaciones comunicarse entre sí. Las API proporcionan un medio para que los desarrolladores accedan a funcionalidades específicas o datos de una aplicación o servicio.
2. **Endpoint:** Es la dirección específica dentro de la API que realiza una función particular. En términos simples, un endpoint es la "dirección" a la que se hace una solicitud para interactuar con un recurso o ejecutar una operación en un servicio web.

Ejemplo:

Imagina una API para la gestión de una biblioteca. Esta API podría tener varios endpoints, como:

- `GET /books``: Este endpoint podría devolver una lista de todos los libros disponibles.
- `POST /books``: Este endpoint podría permitir agregar un nuevo libro a la biblioteca.
- `GET /books/{id}``: Este endpoint podría devolver los detalles de un libro específico, identificado por su ID.
- `PUT /books/{id}``: Este endpoint podría permitir actualizar la información de un libro específico.
- `DELETE /books/{id}``: Este endpoint podría eliminar un libro específico de la base de datos.

Componentes de un Endpoint:

- **Método HTTP:** Como `GET``, `POST``, `PUT``, `DELETE``, que define el tipo de operación que se está realizando.
- **URL:** La dirección que incluye el nombre del recurso y a veces parámetros para especificar el recurso exacto, como `/books`` o `/books/{id}``.
- **Parámetros:** A veces, los endpoints incluyen parámetros en la URL o en el cuerpo de la solicitud para filtrar o especificar detalles adicionales.

Relación entre API y Endpoints:

La API es el conjunto completo de endpoints que permiten la interacción con un servicio. Cada endpoint proporciona acceso a una función específica o conjunto de datos dentro de ese servicio.



que es una api



Una **API** (Interfaz de Programación de Aplicaciones, por sus siglas en inglés: Application Programming Interface) es un conjunto de definiciones y protocolos que permite que una aplicación o servicio interactúe con otro software, facilitando el intercambio de datos y funcionalidades. Las API permiten que diferentes sistemas o componentes de software se comuniquen entre sí de manera estructurada y predefinida.

Características principales de una API:

1. **Interfaz de Comunicación:** Proporciona una manera estándar para que dos aplicaciones se comuniquen entre sí, generalmente mediante el envío de solicitudes y la recepción de respuestas.
2. **Abstracción:** Las API permiten a los desarrolladores usar funcionalidades complejas sin necesidad de entender los detalles internos del sistema. Por ejemplo, un desarrollador puede usar una API de un servicio de pagos para procesar transacciones sin saber cómo funciona el procesamiento de pagos internamente.
3. **Definiciones de Métodos:** Las API definen una serie de métodos o funciones que pueden ser llamados por otras aplicaciones. Estos métodos suelen estar asociados a acciones específicas, como crear, leer, actualizar o eliminar datos.
4. **Tipos de API:**
 - **APIs de Sistema Operativo:** Permiten que las aplicaciones interactúen con los servicios del sistema operativo, como la gestión de archivos o el acceso a recursos de hardware.
 - **APIs de Bibliotecas o Frameworks:** Proporcionan funciones reutilizables para tareas comunes, como manejo de gráficos, acceso a bases de datos, etc.
 - **APIs Web:** Permiten la comunicación entre diferentes aplicaciones a través de la web, a menudo utilizando HTTP/HTTPS. Estas son muy comunes en el desarrollo de aplicaciones web y móviles.
5. **Formatos de Datos:**
 - Las API web generalmente intercambian datos en formatos estándar como JSON (JavaScript Object Notation) o XML (eXtensible Markup Language).
 - En una API RESTful, por ejemplo, los datos pueden ser enviados y recibidos en formato JSON para facilitar su lectura y procesamiento.
6. **Seguridad:** Muchas API implementan mecanismos de autenticación y autorización para asegurar que solo usuarios o aplicaciones autorizadas puedan acceder a sus funcionalidades.

Ejemplo de uso:

Supongamos que tienes una aplicación de clima que necesita mostrar el pronóstico del tiempo. En lugar de recopilar y procesar los datos meteorológicos directamente, la aplicación podría hacer uso de una API de un servicio meteorológico. La aplicación enviaría una solicitud a un endpoint específico de la API, como ``https://api.weather.com/forecast``, y recibiría una respuesta con los datos del pronóstico que luego se mostrarían en la interfaz de usuario.

En resumen, una API es una herramienta esencial que permite la integración y colaboración entre diferentes piezas de software, facilitando la creación de aplicaciones más potentes y flexibles al aprovechar las capacidades de otros servicios y sistemas.



Que es un resource cuando hablamos de endpoint



Cuando hablamos de un **resource** (recurso) en el contexto de un endpoint de una API, nos referimos a una entidad o conjunto de datos que la API maneja y con la cual se pueden realizar diferentes operaciones a través de las solicitudes HTTP. Un recurso es esencialmente el objeto o dato que está siendo accedido, manipulado o consultado mediante la API.

Ejemplo de un Resource:

Imaginemos una API que gestiona una biblioteca digital. En este caso, algunos recursos típicos podrían ser:

- ``books`` (**libros**): Representa el conjunto de todos los libros disponibles en la biblioteca.
- ``authors`` (**autores**): Representa el conjunto de autores cuyos libros están en la biblioteca.
- ``users`` (**usuarios**): Representa las personas que tienen acceso a la biblioteca.

Recursos y Endpoints:

Un **endpoint** es una dirección URL específica que apunta a un recurso o conjunto de recursos en la API. Los recursos suelen representarse como sustantivos en la URL de un endpoint.

Por ejemplo:

1. Lista de todos los libros:

- **Endpoint:** ``GET /books``
- **Resource:** ``books`` (todos los libros disponibles en la biblioteca)

2. Detalles de un libro específico:

- **Endpoint:** ``GET /books/{id}``
- **Resource:** ``book`` (un libro en particular, identificado por su ID)

3. Crear un nuevo libro:

- **Endpoint:** ``POST /books``
- **Resource:** ``book`` (el nuevo libro que se va a añadir)

4. Actualizar un libro existente:

- **Endpoint:** `PUT /books/{id}`
- **Resource:** `book` (el libro que se va a actualizar, identificado por su ID)

5. Eliminar un libro:

- **Endpoint:** `DELETE /books/{id}`
- **Resource:** `book` (el libro que se va a eliminar, identificado por su ID)

Relación entre Recursos y CRUD:

En la mayoría de las APIs RESTful, los recursos están asociados a las operaciones CRUD (Crear, Leer, Actualizar, Eliminar):

- **Crear:** `POST /books` (crea un nuevo recurso de tipo `book`)
- **Leer:** `GET /books` (lee o recupera una lista de recursos de tipo `book`)
- **Leer:** `GET /books/{id}` (lee o recupera un recurso específico de tipo `book`)
- **Actualizar:** `PUT /books/{id}` (actualiza un recurso de tipo `book`)
- **Eliminar:** `DELETE /books/{id}` (elimina un recurso de tipo `book`)

Importancia de los Recursos en APIs:

Definir recursos claramente es fundamental para diseñar una API que sea intuitiva y fácil de usar. Cada recurso debe representarse de forma clara y sus endpoints deben reflejar las operaciones posibles sobre esos recursos. Esto permite a los desarrolladores entender fácilmente cómo interactuar con la API y qué esperar de sus respuestas.



Que son los metodos web cuando se hace un request o un response



Los **métodos web** o **métodos HTTP** son un conjunto de comandos que especifican la acción que se desea realizar en un recurso cuando se hace una solicitud (request) a una API. Estos métodos definen qué tipo de operación se ejecuta sobre un recurso (por ejemplo, crear, leer, actualizar o eliminar) y forman parte de los protocolos HTTP y HTTPS que se utilizan para la comunicación en la web.

Principales Métodos HTTP:

1. GET:

- **Propósito:** Obtener (leer) datos de un recurso.
- **Uso común:** Recuperar información de un servidor, como la lista de todos los usuarios o los detalles de un artículo específico.
- **Ejemplo:** `GET /books` (obtener una lista de todos los libros)
- **Características:** Es un método seguro e idempotente, lo que significa que no modifica el estado del recurso y realizar la misma solicitud varias veces no debería tener efectos

secundarios.

2. POST:

- **Propósito:** Crear un nuevo recurso en el servidor.
- **Uso común:** Enviar datos al servidor para crear un nuevo registro, como añadir un nuevo libro a la base de datos.
- **Ejemplo:** `POST /books`` (crear un nuevo libro)
- **Características:** No es idempotente; realizar la misma solicitud varias veces podría resultar en la creación de múltiples recursos.

3. PUT:

- **Propósito:** Actualizar o reemplazar un recurso existente.
- **Uso común:** Modificar un recurso existente, como actualizar la información de un libro específico.
- **Ejemplo:** `PUT /books/{id}`` (actualizar los detalles de un libro con un ID específico)
- **Características:** Es idempotente; realizar la misma solicitud varias veces tendrá el mismo efecto que hacerla una sola vez.

4. DELETE:

- **Propósito:** Eliminar un recurso del servidor.
- **Uso común:** Borrar un recurso específico, como eliminar un libro de la base de datos.
- **Ejemplo:** `DELETE /books/{id}`` (eliminar un libro con un ID específico)
- **Características:** Es idempotente; realizar la misma solicitud varias veces no cambia el estado después de la primera eliminación.

5. PATCH:

- **Propósito:** Aplicar modificaciones parciales a un recurso existente.
- **Uso común:** Actualizar solo algunos campos de un recurso, como modificar el título de un libro sin cambiar el resto de la información.
- **Ejemplo:** `PATCH /books/{id}`` (modificar parcialmente los detalles de un libro con un ID específico)
- **Características:** No es necesariamente idempotente; su comportamiento depende de cómo se implemente.

6. OPTIONS:

- **Propósito:** Describir las opciones de comunicación disponibles para el recurso especificado.
- **Uso común:** Consultar el servidor sobre los métodos HTTP que admite para un recurso específico.
- **Ejemplo:** `OPTIONS /books`` (obtener los métodos permitidos para el recurso ``books``)
- **Características:** No modifica el estado del recurso; es utilizado principalmente para comprobar la configuración de CORS (Cross-Origin Resource Sharing).

Relación entre Métodos HTTP, Requests, y Responses:

- **Request (Solicitud):** Cuando un cliente (por ejemplo, un navegador web o una aplicación) quiere interactuar con un recurso en un servidor, envía una solicitud HTTP. Esta solicitud incluye un método HTTP que indica la acción deseada, una URL que especifica el recurso, y opcionalmente un cuerpo de datos (en métodos como POST o PUT).

- **Response (Respuesta):** El servidor procesa la solicitud y devuelve una respuesta HTTP. Esta respuesta incluye un código de estado HTTP que indica si la operación fue exitosa, junto con cualquier dato solicitado (en el caso de GET) o un mensaje confirmando la acción (en el caso de POST, PUT, DELETE).

Ejemplo de un Flujo de Solicitud y Respuesta:

1. Solicitud:

- Método: `GET`
- URL: `https://api.example.com/users/123`
- Acción: Obtener la información del usuario con ID 123.

2. Respuesta:

- Código de estado: `200 OK`
- Cuerpo: `{ "id": 123, "name": "John Doe", "email": "john@example.com" }`
- Acción: El servidor devuelve los detalles del usuario solicitado.

Estos métodos forman la base de la interacción con los recursos de una API, permitiendo a los desarrolladores construir aplicaciones que puedan realizar operaciones de lectura, escritura, actualización y eliminación de datos de manera estructurada y eficiente.



Ayúdame a crear un programa en Python para consultar una tabla de mockapi.io en la URL <https://66b4e5549f9169621ea4c38c.mockapi.io/api/v1/contactos>, que me muestre todos los registros en JSON formateado, en formato Dataframe y los exporte a un csv



Memoria actualizada

Aquí tienes un programa en Python que consulta la tabla en MockAPI, muestra los registros en JSON formateado, los convierte a un DataFrame y luego los exporta a un archivo CSV:

python

```
import requests
import pandas as pd
import json

# URL de la API
url = "https://66b4e5549f9169621ea4c38c.mockapi.io/api/v1/contactos"

# Realizar la solicitud GET a la API
response = requests.get(url)

# Comprobar si la solicitud fue exitosa
if response.status_code == 200:
    # Obtener los datos en formato JSON
    contactos_json = response.json()

    # Mostrar los registros en formato JSON formateado
```

```
print("Registros en formato JSON formateado:")
print(json.dumps(contactos_json, indent=4))

# Convertir los datos a un DataFrame de pandas
df_contactos = pd.DataFrame(contactos_json)

# Mostrar el DataFrame
print("\nRegistros en formato DataFrame:")
print(df_contactos)

# Exportar el DataFrame a un archivo CSV
df_contactos.to_csv("contactos.csv", index=False)
print("\nLos datos han sido exportados a 'contactos.csv'")
else:
    print(f"Error al consultar la API: {response.status_code}")
```

Desglose del código:

1. Importación de módulos:

- ``requests``: Para hacer la solicitud HTTP a la API.
- ``pandas``: Para manipular y analizar datos, así como convertir el JSON en un DataFrame.
- ``json``: Para formatear y mostrar los datos JSON.

2. Consulta de la API:

- Se hace una solicitud GET a la URL proporcionada.

3. Verificación de la respuesta:

- Si la solicitud es exitosa (código 200), se procesan los datos.

4. Mostrar datos en JSON:

- Se formatea y se imprime el JSON para facilitar su lectura.

5. Convertir a DataFrame:

- Se convierte el JSON en un DataFrame de pandas, que es una estructura de datos tabular.

6. Exportar a CSV:

- El DataFrame se exporta a un archivo CSV llamado ``contactos.csv``.

Ejecución del código:

Guarda este código en un archivo Python (``consultar_contactos.py``, por ejemplo) y ejecútalo. Si todo está configurado correctamente, verás los datos de la tabla en la terminal, y se creará un archivo ``contactos.csv`` en el mismo directorio donde se ejecuta el script.

Si necesitas alguna modificación o ayuda adicional, ¡avísame!



Ahora, ayúdame con el programa para mostrar solamente un determinado registro de la misma URL de MockAPI, muestra todos los campos en formato plano y ya no lo muestres en Dataframe ni lo exportes a CSV



Aquí tienes un programa en Python que consulta un registro específico en la URL de MockAPI y lo muestra en formato plano:

python

```
import requests

# URL base de la API
url_base = "https://66b4e5549f9169621ea4c38c.mockapi.io/api/v1/contactos"

# ID del registro que deseas consultar
registro_id = "1" # Cambia "1" por el ID del registro que quieres consultar

# Construir la URL para consultar un registro específico
url_registro = f"{url_base}/{registro_id}"

# Realizar la solicitud GET a la API
response = requests.get(url_registro)

# Comprobar si la solicitud fue exitosa
if response.status_code == 200:
    # Obtener los datos del registro en formato JSON
    registro_json = response.json()

    # Mostrar todos los campos en formato plano
    print("Registro en formato plano:")
    for key, value in registro_json.items():
        print(f"{key}: {value}")
else:
    print(f"Error al consultar la API: {response.status_code}")
```

Desglose del código:

1. URL base y ID del registro:

- `url_base`: La URL base de la API.
- `registro_id`: El ID del registro que deseas consultar. Puedes cambiar `"1"` por el ID del registro que te interese.

2. Construcción de la URL:

- Se concatena la URL base con el ID del registro para obtener la URL específica del recurso.

3. Consulta de la API:

- Se realiza una solicitud GET a la URL específica del registro.

4. Verificación de la respuesta:

- Si la solicitud es exitosa (código 200), se procesan y muestran los datos del registro.

5. Mostrar campos en formato plano:

- Se recorre el JSON y se imprime cada campo en formato `clave: valor`.

Ejecución del código:

Guarda este código en un archivo Python (`consultar_registro.py`, por ejemplo) y ejecútalo. El script mostrará los datos del registro específico en formato plano en la terminal.

Si necesitas consultar un registro diferente, solo cambia el valor de `registro_id`. ¡Dime si hay algo más en lo que te pueda ayudar!

Ahora dame el programa para agregar un registro en la misma URL de MockAPI



Aquí tienes un programa en Python que agrega un nuevo registro a la tabla en MockAPI usando una solicitud `POST`:

python

```
import requests

# URL de la API
url = "https://66b4e5549f9169621ea4c38c.mockapi.io/api/v1/contactos"

# Datos del nuevo registro que deseas agregar
nuevo_registro = {
    "nombre": "Juan Pérez",
    "email": "juan.perez@example.com",
    "telefono": "+52 123 456 7890",
    "direccion": "Calle Falsa 123, Ciudad de México"
}

# Realizar la solicitud POST para agregar el nuevo registro
response = requests.post(url, json=nuevo_registro)

# Comprobar si la solicitud fue exitosa
if response.status_code == 201:
    print("Registro agregado exitosamente:")
    print(response.json()) # Mostrar el registro agregado
else:
    print(f"Error al agregar el registro: {response.status_code}")
```

Desglose del código:

1. URL de la API:

- La URL apunta a la tabla de contactos donde deseas agregar el nuevo registro.

2. Datos del nuevo registro:

- `nuevo_registro`: Un diccionario con los campos y valores que quieres agregar al nuevo contacto. Puedes ajustar los campos y valores según las necesidades.

3. Solicitud POST:

- Se realiza una solicitud `POST` a la URL de la API, enviando el `nuevo_registro` en formato JSON.

4. Verificación de la respuesta:

- Si la solicitud es exitosa (código 201), se imprime el registro agregado. Si falla, se muestra un mensaje de error con el código de estado.

Ejecución del código:

Guarda este código en un archivo Python (`agregar_registro.py`, por ejemplo) y ejecútalo. El script agregará el nuevo registro a la tabla en MockAPI y mostrará los datos del registro que se ha añadido.

Si necesitas modificar los datos del registro, simplemente ajusta el diccionario `nuevo_registro`. ¡Dime si hay algo más en lo que te pueda ayudar!



Ahora dame el programa para modificar un registro en la misma URL de MockAPI



Aquí tienes un programa en Python que modifica un registro existente en MockAPI usando una solicitud `PUT`:

python

```
import requests

# URL base de la API
url_base = "https://66b4e5549f9169621ea4c38c.mockapi.io/api/v1/contactos"

# ID del registro que deseas modificar
registro_id = "1" # Cambia "1" por el ID del registro que quieres modificar

# URL completa para el registro específico
url_registro = f"{url_base}/{registro_id}"

# Datos que deseas actualizar en el registro
registro_modificado = {
    "nombre": "Juan Pérez Modificado",
    "email": "juan.perez.modificado@example.com",
    "telefono": "+52 987 654 3210",
    "direccion": "Avenida Siempre Viva 742, Ciudad de México"
}

# Realizar la solicitud PUT para modificar el registro
response = requests.put(url_registro, json=registro_modificado)

# Comprobar si la solicitud fue exitosa
if response.status_code == 200:
    print("Registro modificado exitosamente:")
    print(response.json()) # Mostrar el registro modificado
else:
    print(f"Error al modificar el registro: {response.status_code}")
```

Desglose del código:

1. URL base y ID del registro:

- `url_base`: La URL base de la API.
- `registro_id`: El ID del registro que deseas modificar. Cambia ``"1"`` por el ID del registro que necesitas actualizar.

2. URL específica para el registro:

- Se construye la URL completa para el registro específico concatenando la URL base y el ID del registro.

3. Datos del registro modificado:

- ``registro_modificado``: Un diccionario con los campos y valores que quieres actualizar en el registro. Puedes ajustar los campos según sea necesario.

4. Solicitud PUT:

- Se realiza una solicitud ``PUT`` a la URL específica del registro, enviando los datos modificados en formato JSON.

5. Verificación de la respuesta:

- Si la solicitud es exitosa (código 200), se imprimen los datos del registro modificado. Si falla, se muestra un mensaje de error con el código de estado.

Ejecución del código:

Guarda este código en un archivo Python (``modificar_registro.py``, por ejemplo) y ejecútalo. El script modificará el registro especificado en MockAPI y mostrará los datos actualizados.

Si necesitas modificar un registro diferente, ajusta el valor de ``registro_id``, y si deseas cambiar otros datos, edita el diccionario ``registro_modificado``. ¡Avísame si hay algo más en lo que te pueda asistir!



Ahora dame el programa para eliminar un registro en la misma URL de MockAPI



Aquí tienes un programa en Python que elimina un registro específico en MockAPI usando una solicitud ``DELETE``:

python

```
import requests

# URL base de la API
url_base = "https://66b4e5549f9169621ea4c38c.mockapi.io/api/v1/contactos"

# ID del registro que deseas eliminar
registro_id = "1" # Cambia "1" por el ID del registro que quieres eliminar

# URL completa para el registro específico
url_registro = f"{url_base}/{registro_id}"

# Realizar la solicitud DELETE para eliminar el registro
response = requests.delete(url_registro)

# Comprobar si la solicitud fue exitosa
if response.status_code == 200:
    print(f"Registro con ID {registro_id} eliminado exitosamente.")
else:
    print(f"Error al eliminar el registro: {response.status_code}")
```

Desglose del código:

1. URL base y ID del registro:

- ``url_base``: La URL base de la API.
- ``registro_id``: El ID del registro que deseas eliminar. Cambia ``"1"`` por el ID del registro que deseas borrar.

2. URL específica para el registro:

- Se construye la URL completa para el registro específico concatenando la URL base y el ID del registro.

3. Solicitud DELETE:

- Se realiza una solicitud ``DELETE`` a la URL específica del registro.

4. Verificación de la respuesta:

- Si la solicitud es exitosa (código 200), se muestra un mensaje confirmando la eliminación. Si falla, se muestra un mensaje de error con el código de estado.

Ejecución del código:

Guarda este código en un archivo Python (``eliminar_registro.py``, por ejemplo) y ejecútalo. El script eliminará el registro especificado en MockAPI.

Si necesitas eliminar un registro diferente, ajusta el valor de ``registro_id``. ¡Dime si necesitas más ayuda!

ChatGPT puede cometer errores. Considera verificar la información importante.