

# **Laboratorio: Análisis Inteligente de Datos desde una API REST usando LINQ y OpenAI en C#**

Laboratorio desarrollado por:

Jesús Rodríguez

José Ortega

Descripción de la API y proceso de integración

La API utilizada en este proyecto fue

<https://jsonplaceholder.typicode.com/posts>, la cual proporciona datos públicos en formato JSON que simulan publicaciones en línea.

Para obtener los datos, se realizó una solicitud HTTP GET utilizando la clase `HttpClient` incluida en .NET.

Luego, la respuesta en formato JSON fue deserializada a una lista de objetos `Post` empleando la librería `Newtonsoft.Json`, instalada mediante NuGet.

Esta integración permitió trabajar con los datos como una colección de objetos en C#, facilitando la aplicación de consultas LINQ para filtrar, ordenar, agrupar y analizar la información.

Se implementó la conexión con la API de OpenAI para generar explicaciones automáticas, resúmenes y clasificaciones de los resultados, combinando análisis lógico con interpretación basada en inteligencia artificial.

Listado de las 20 consultas LINQ implementadas

1) Filtrar los posts cuyo UserId sea igual a 1.

Explicación: Permite obtener solo las publicaciones pertenecientes al usuario con ID 1.

2) Ordenar los posts por título alfabéticamente y mostrar los primeros 5.

Explicación: Facilita la visualización de los primeros títulos ordenados de forma alfabética.

3) Filtrar los títulos que contienen la palabra 'qui'.

Explicación: Ayuda a encontrar publicaciones que contengan una palabra clave dentro del título.

4) Mostrar los posts cuyo cuerpo tenga más de 100 caracteres.

Explicación: Identifica las publicaciones con cuerpos de texto extensos.

5) Agrupar los posts por UserId y contar cuántos posts tiene cada usuario.

Explicación: Muestra la distribución de publicaciones por usuario mediante un conteo agrupado.

6) Obtener el Id máximo y mínimo entre todos los posts.

Explicación: Permite conocer los valores extremos del campo Id.

7) Calcular la longitud promedio del cuerpo (body) de los posts.

Explicación: Da una idea del tamaño promedio de las descripciones o cuerpos de texto.

8) Mostrar los posts con Id par, tomando solo los primeros 10.

Explicación: Filtra publicaciones con identificadores pares para un muestreo rápido.

9) Concatenar los primeros 10 títulos de posts en una sola cadena.

Explicación: Genera una cadena compacta con los primeros títulos, útil para resúmenes.

10) Verificar si existe algún título vacío o nulo.

Explicación: Verifica la calidad de los datos, asegurando que todos los títulos estén completos.

11) Mostrar los últimos 5 posts ordenados por Id descendente.

Explicación: Permite observar las últimas publicaciones ingresadas en la lista.

12) Contar los posts cuyo título contiene la palabra 'est'.

Explicación: Busca coincidencias con una palabra clave ('est') para análisis de texto.

13) Mostrar los usuarios que tienen más de 8 posts.

Explicación: Detecta qué usuarios publican con mayor frecuencia.

14) Listar los tres títulos más largos por cada usuario.

Explicación: Permite analizar los títulos más extensos de cada usuario, relacionados con su estilo.

15) Obtener los 10 títulos distintos (sin repetir) de los posts.

Explicación: Evita duplicados mostrando títulos únicos.

16) Agrupar los posts según si su Id es par o impar.

Explicación: Clasifica las publicaciones según su número de Id, separando pares e impares.

17) Filtrar los posts de un usuario específico que contengan cierta palabra clave en el título o cuerpo.

Explicación: Combina filtros por usuario y palabra clave para búsquedas específicas.

18) Calcular el promedio global de longitud del cuerpo de todos los posts.

Explicación: Obtiene una medida global del tamaño promedio del contenido.

19) Calcular la proporción entre posts largos ( $\geq 50$  caracteres) y cortos ( $< 50$  caracteres).

Explicación: Mide la proporción entre publicaciones largas y cortas, útil para análisis estadístico.

20) Saltar los primeros 10 posts y tomar los siguientes 5 para mostrar.

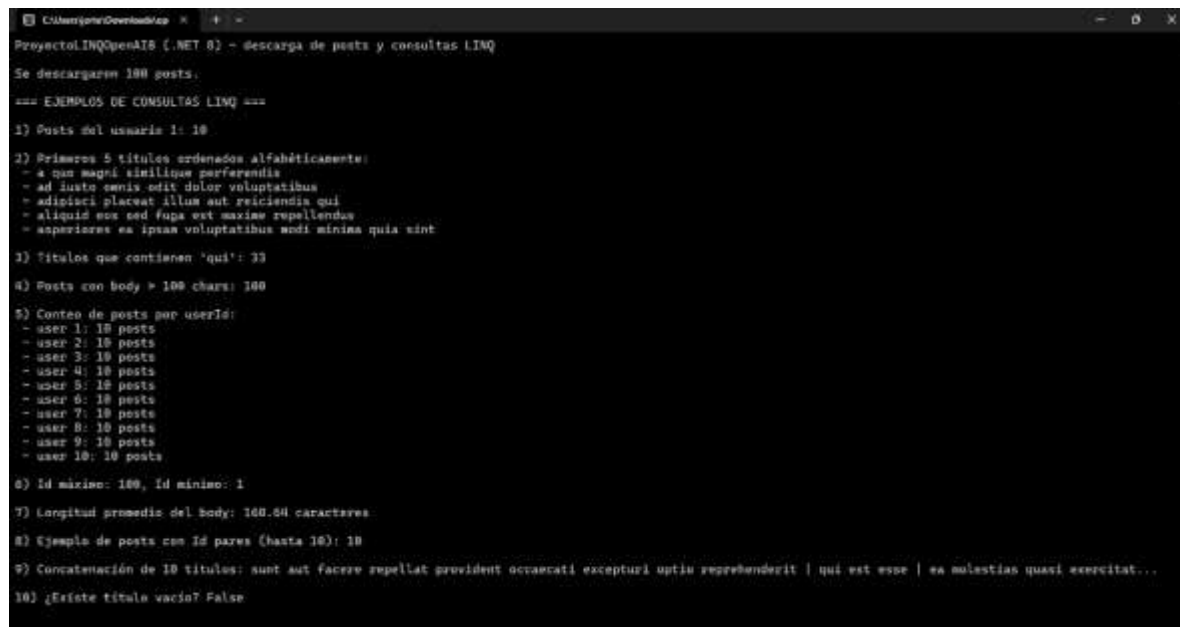
Explicación: Muestra un subconjunto específico de publicaciones tras omitir las primeras, útil para paginación.

Capturas de pantalla de resultados

Se incluyeron imágenes de la consola mostrando la ejecución de cada consulta y los resultados obtenidos (no incluidas en este documento por formato de texto).

Capturas de los resultados de consultas LINQ:

1.



```

C:\Programas\VisualStudio\ - + -
ProyectoLINQOpenAPI (.NET 8) - descarga de posts y consultas LINQ
Se descargaron 100 posts.
=== EJEMPLOS DE CONSULTAS LINQ ===
1) Posts del usuario 1: 10
2) Primeros 5 títulos ordenados alfabéticamente:
- a qui magna similique perferendis
- ad iusto enim edit dolor voluptatibus
- adipisci placeat illum aut reiciendis qui
- aliquis non sed fupa est maxime repellendus
- asperiores ea ipsam voluptatibus modi minima quia sint
3) Títulos que contienen 'qui': 33
4) Posts con body > 100 chars: 100
5) Conteo de posts por userId:
- user 1: 10 posts
- user 2: 10 posts
- user 3: 10 posts
- user 4: 10 posts
- user 5: 10 posts
- user 6: 10 posts
- user 7: 10 posts
- user 8: 10 posts
- user 9: 10 posts
- user 10: 10 posts
6) Id máximo: 100, Id mínimo: 1
7) Longitud promedio del body: 160.54 caracteres
8) Ejemplo de posts con Id pares (hasta 10): 10
9) Concatenación de 10 títulos: sunt aut facere repellat provident occaecati excepturi optio reprehenderit | qui est esse | ea molestias quasi exercitation...
10) ¿Existe título vacío? False

```

2.

```

11) Últimas 5 posts (Id, Title):
- 100: at nam consequat ea labore ea harum
- 99: temporibus sit alias delectus eligendi possimus magni
- 98: laboribus dolor voluptates
- 97: quas fugiat ut perspiciatis vero provident
- 96: quærat velit veniam amet cupiditate aut namquam ut sequi

12) Títulos que contienen 'est': 10

13) Usuarios con mas de 8 posts
Usuario 1 tiene 10 posts
Usuario 2 tiene 10 posts
Usuario 3 tiene 10 posts
Usuario 4 tiene 10 posts
Usuario 5 tiene 10 posts
Usuario 6 tiene 10 posts
Usuario 7 tiene 10 posts
Usuario 8 tiene 10 posts
Usuario 9 tiene 10 posts
Usuario 10 tiene 10 posts

14) Títulos mas largos de cada usuario
Usuario 1:
- sunt aut facere repellat provident occaecati excepturi optio reprehenderit
- ea molestias quasi exercitationem repellat qui ipsa sit aut
- nesciunt iure remis dolorum tempora et accusantium
Usuario 2:
- sint suscipit perspiciatis velit dolorum rerum ipsa laboribus odio
- dolorum ut in voluptas mollitia et saepe quo animi
- fugit voluptas sed molestias voluptates provident
Usuario 3:
- asperiores ea ipsam voluptatibus modi minima quia sint
- delectus alias et corporis nulla voluptas sequi
- dolore sint quo a velit explicabo quia nam
Usuario 4:
- explicabo et eos deleniti nostrum ab id repellendus
- id nihil consequat molestias animi provident
- fuga nam accusamus voluptas reiciendis itaque
Usuario 5:
- repellendus qui recusandae incidunt voluptates tenetur qui omnis exercitationem

```

3.

```

Usuario 6:
- repellendus qui recusandae incidunt voluptates tenetur qui omnis exercitationem
- eligendi iste nostrum consequatur ad ipsam sit aut
- comodi ullam sint et excepturi error explicabo praesentium voluptas
Usuario 7:
- consequat placeat enim quisquam quia reprehenderit fugit veritatis facere
- qui enim et consequatur quia animi quia voluptate quibusdam
- soluta aliquam aperiam consequatur illo quia voluptas
Usuario 8:
- voluptas blanditiis repellendus animi ducimus error sapiente et suscipit
- voluptates doloribus consectetur est ut ducimus
- odio quis facere architecto reiciendis optio
Usuario 9:
- consequatur deleniti eos quia temporibus ab aliquid at
- enim unde ratione doloribus quas enim ut sit sapiente
- pariatur consequatur quia magnam autem enim non amet
Usuario 10:
- optio ipsam molestias necessitatibus occaecati facilis veritatis dolores aut
- dolore veritatis porro provident adipisci blanditiis et sunt
- tempora cum veritatis voluptas quo dolores vero
Usuario 11:
- quærat velit veniam amet cupiditate aut namquam ut sequi
- temporibus sit alias delectus eligendi possimus magni
- quas fugiat ut perspiciatis vero provident

15) 30 Títulos distintos:
sunt aut facere repellat provident occaecati excepturi optio reprehenderit
qui est enim
ea molestias quasi exercitationem repellat qui ipsa sit aut
sunt et est occaecati
nesciunt quas odio
dolorem esse magni enim aperiam quia
magnam facilis autem
dolorem dolore est ipsam
nesciunt iure omnis dolorem tempora et accusantium
optio molestias id quia sunt

```

4.

```

16) Agrupar 10 en grupos por el valor
ID: 100:
sunt aut facere repellat provident occaecati excepturi optio reprehenderit (ID: 1)
ea molestias quasi exercitationem repellat qui ipsa sit aut (ID: 2)
nesciunt iure omnis dolorem tempora et accusantium (ID: 3)
magnam facilis autem (ID: 4)
nesciunt iure omnis dolorem tempora et accusantium (ID: 5)
et ea vero quia laudantium autem (ID: 6)
dolorem ut in voluptas mollitia et saepe quo animi (ID: 7)
explicabo quod voluptas (ID: 8)
fugit voluptas sed molestias voluptates provident (ID: 9)
adipisci placeat illum aut reiciendis qui (ID: 10)
asperiores ea ipsam voluptatibus modi minima quia sint (ID: 11)
velit id sita nihil remque (ID: 12)
cum alias distinctio quo odio (ID: 13)
quasi id et eos tenetur aut quo autem (ID: 14)
ipsum enim eum necessitatibus culpa ea (ID: 15)
illum ut sedibus id nec vel, diamandum (ID: 16)
qui voluptate molestias dolorum (ID: 17)
id nihil consequatur molestias animi provident (ID: 18)
provident vel et sit exilise est (ID: 19)
eos dolorem iste accusantium est eaque quam (ID: 20)
nam est facere (ID: 21)
eligendi iste nostrum consequatur adipisci praesentium sit beatae perferendis (ID: 22)
et remque praesentium omnis eius laborum laudantium iure (ID: 23)
quibusdam tempore rem aut consequat (ID: 24)
Laborum rem aut aut ut assumenda perspiciatis voluptas (ID: 25)
soluta aliquam aperiam consequatur illo quia voluptas (ID: 26)
et quo aut faciam alias (ID: 27)
aut vel voluptatem et non libero (ID: 28)
sed id est est (ID: 29)
qui comodi nihil ut voluptas et quia id accusantium (ID: 30)
voluptates doloribus consectetur est ut ducimus (ID: 31)
voluptas blanditiis repellendus animi ducimus error sapiente et suscipit (ID: 32)
consequatur id enim sunt et et (ID: 33)
aliquid eos sed fuga est maxime repellendus (ID: 34)
fugiat quod pariatur aut ducimus (ID: 35)
et iusto veritas et illum aut fuga (ID: 36)
consequatur deleniti enim quia temporibus ab aliquid at (ID: 37)
dignissimos eum dolor ut enim et delectus in (ID: 38)

```

[illegible][illegible]

1.

```
-- consulta con ORDER BY --
```

**Consulta:**  
`posts.Where(p => p.UserId == 1).Take(5);`  
**Explicación:**  
Esta consulta LINQ filtra una colección llamada `posts` para obtener solo aquellos elementos (o publicaciones) cuyo `UserId` es igual a 1. En otras palabras, está buscando solo las publicaciones que pertenecen al usuario con el identificador 1. Luego, convierte el resultado filtrado en una lista utilizando el método `Take(5)`. Así que al final, obtienes una lista de publicaciones que son de ese usuario específico.

**Consulta:**  
`posts.OrderBy(p => p.Title).Take(5);`  
**Explicación:**  
La consulta LINQ `posts.OrderBy(p => p.Title).Take(5);` realiza lo siguiente:

1. **Ordenar:** La colección de `posts` se ordena del título de cada post (`p.Title`). Esto significa que las posts se organizarán alfabéticamente según sus títulos.
2. **Tomar:** Las primeras 5 elementos de la lista ordenada. Es decir, después de ordenar, selecciona solo los cinco primeros posts.

En resumen, esta consulta devuelve las cinco posts con los títulos más bajos en orden alfabético.

**Consulta:**  
`posts.Where(p => p.Body.Length > 100);`  
**Explicación:**  
La consulta LINQ `posts.Where(p => p.Body.Length > 100)` filtra una colección de objetos llamada `posts`. En este caso, selecciona solo aquellos elementos (o publicaciones) cuya propiedad `Body` tiene una longitud mayor a 100 caracteres. Es decir, devuelve una nueva colección que contiene únicamente las publicaciones que cumplen con esta condición.

**Consulta:**  
`posts.GroupBy(p => p.UserId).Select(g => new { p.Key, Count = g.Count() });`  
**Explicación:**  
Esta consulta LINQ agrupa una colección de publicaciones (`posts`) por el identificador del usuario (`UserId`) que las creó. Luego, para cada grupo de publicaciones que pertenecen a un mismo usuario, crea un nuevo objeto anónimo que contiene dos propiedades:

1. `Key`: que representa al `UserId` del usuario.
2. `Count`: que es el número total de publicaciones que ese usuario ha creado.

En resumen, la consulta devuelve una lista de usuarios junto con la cantidad de publicaciones que cada uno ha realizado.

2.

Resumen de resultados:  
 Todos los usuarios enumerados tienen la misma cantidad de publicaciones, con un total de 10 posts cada uno. En total, hay 10 usuarios. Esto sugiere una distribución uniforme de contenido entre los usuarios.

Clasificación de posts:  
 Aquí tienes una clasificación de los posts en temas o categorías generales:

```
### 1. **Responsabilidad y Consecuencias**
- **Post 1:** "sunt aut facere repellat provident occaecati excepturi optio reprehenderit: quia et suscipit..."
- **Post 2:** "qui est esse: est rerum tempore vitae..."
- **Post 3:** "ea molestias quasi exercitationem repellat qui ipsa sit aut: et iusto sed que iure..."
- **Post 4:** "cum et est occaecati: allan et saepe reiciendis voluptatem adipisci..."
- **Post 5:** "nesciunt quas odio: repudiandae veniam quaeat sunt sed..."

### 2. **Problemas y Soluciones**
- **Post 6:** "dolores eum magni eos aperiam quia: ut aspernatur corporis harum nihil quis provident sequi..."
- **Post 7:** "magnas facilis ante: dolore placeat quibusdam ea quo vitae..."
- **Post 8:** "dolores dolore est ipsam: dignissimos aperiam dolorem qui eum..."

### 3. **Justicia y Verdad**
- **Post 9:** "nesciunt iure omnis dolorem tempora et accusantium: consectetur animi nesciunt iure dolorem..."
- **Post 10:** "optio molestias id quia eum: quo et expedita modi cum officia vel magni..."

### 4. **Percepción y Realidad**
- **Post 11:** "et ea vero quia laudantium autem: delectis reiciendis molestia occaecati non minima..."
- **Post 12:** "in quibusdam tempore odit est dolorem: itaque id aut magnam..."

### 5. **Emociones y Experiencias**
- **Post 13:** "dolorem et in voluptate mollitia et saepe quo animi: aut dicta possim sint mollitia..."
- **Post 14:** "voluptatem eligendi optio: fuga et accusamus dolorum perferendis illo voluptas..."

### 6. **Desafíos y Obstáculos**
- **Post 15:** "eveniet quod temporibus: reprehenderit quos placeat..."
```

Esta clasificación es general y puede variar dependiendo del enfoque que se desee dar a cada post. Si necesitas una clasificación más específica o detallada, házmelo saber.

Consultas generadas por OpenAI:  
 Claro, aquí tienes tres consultas LINQ interesantes utilizando el modelo de datos 'Post'. Asumiremos que tienes una lista de objetos 'Post' llamada 'posts'.

3.

Resumen de resultados:  
 Todos los usuarios enumerados tienen la misma cantidad de publicaciones, con un total de 10 posts cada uno. En total, hay 10 usuarios. Esto sugiere una distribución uniforme de contenido entre los usuarios.

Clasificación de posts:  
 Aquí tienes una clasificación de los posts en temas o categorías generales:

```
### 1. **Responsabilidad y Consecuencias**
- **Post 1:** "sunt aut facere repellat provident occaecati excepturi optio reprehenderit: quia et suscipit..."
- **Post 2:** "qui est esse: est rerum tempore vitae..."
- **Post 3:** "ea molestias quasi exercitationem repellat qui ipsa sit aut: et iusto sed que iure..."
- **Post 4:** "cum et est occaecati: allan et saepe reiciendis voluptatem adipisci..."
- **Post 5:** "nesciunt quas odio: repudiandae veniam quaeat sunt sed..."

### 2. **Problemas y Soluciones**
- **Post 6:** "dolores eum magni eos aperiam quia: ut aspernatur corporis harum nihil quis provident sequi..."
- **Post 7:** "magnas facilis ante: dolore placeat quibusdam ea quo vitae..."
- **Post 8:** "dolores dolore est ipsam: dignissimos aperiam dolorem qui eum..."

### 3. **Justicia y Verdad**
- **Post 9:** "nesciunt iure omnis dolorem tempora et accusantium: consectetur animi nesciunt iure dolorem..."
- **Post 10:** "optio molestias id quia eum: quo et expedita modi cum officia vel magni..."

### 4. **Percepción y Realidad**
- **Post 11:** "et ea vero quia laudantium autem: delectis reiciendis molestia occaecati non minima..."
- **Post 12:** "in quibusdam tempore odit est dolorem: itaque id aut magnam..."

### 5. **Emociones y Experiencias**
- **Post 13:** "dolorem et in voluptate mollitia et saepe quo animi: aut dicta possim sint mollitia..."
- **Post 14:** "voluptatem eligendi optio: fuga et accusamus dolorum perferendis illo voluptas..."

### 6. **Desafíos y Obstáculos**
- **Post 15:** "eveniet quod temporibus: reprehenderit quos placeat..."
```

Esta clasificación es general y puede variar dependiendo del enfoque que se desee dar a cada post. Si necesitas una clasificación más específica o detallada, házmelo saber.

Consultas generadas por OpenAI:  
 Claro, aquí tienes tres consultas LINQ interesantes utilizando el modelo de datos 'Post'. Asumiremos que tienes una lista de objetos 'Post' llamada 'posts'.

4.

Consultas generadas por OpenAI:  
 Claro, aquí tienes tres consultas LINQ interesantes utilizando el modelo de datos 'Post'. Asumiremos que tienes una lista de objetos 'Post' llamada 'posts'.

```
### 1. Obtener los títulos de los posts de un usuario específico

'''csharp
int userId = 1; // Cambia este valor según el usuario que quieras filtrar
var titulosDeUsuario = posts
    .Where(post => post.UserId == userId)
    .Select(post => post.Title)
    .ToList();
'''

**Explicación:** Esta consulta filtra los posts para obtener solo aquellos que pertenecen a un usuario específico (en este caso, el usuario con 'UserId' igual a 1). Luego, selecciona solo los títulos de esos posts y los convierte en una lista.

### 2. Contar el número de posts por usuario

'''csharp
var countsPorUsuario = posts
    .GroupBy(post => post.UserId)
    .Select(group => new
    {
        UserId = group.Key,
        CantidadPosts = group.Count()
    })
    .ToList();
'''

**Explicación:** Aquí agrupamos los posts por 'UserId' y luego contamos cuántos posts hay para cada usuario. La consulta devuelve una lista de objetos anónimos que contienen el 'UserId' y la cantidad de posts asociados a ese usuario.
```

5.

```
## 3. Obtener los posts que contienen una palabra clave en el título

'''csharp
string palabraClave = "ejemplo"; // Cambia esto según la palabra clave que quieras buscar
var postsConPalabraClave = posts
    .Where(post => post.Title.Contains(palabraClave, StringComparison.OrdinalIgnoreCase))
    .ToList();
'''

**Explicación:** Esta consulta busca todos los posts cuyos títulos contienen una palabra clave específica (en este caso, "ejemplo"). Utiliza 'StringComparison.OrdinalIgnoreCase' para que la búsqueda no distinga entre mayúsculas y minúsculas. Los resultados se almacenan en una lista.

Estas consultas son ejemplos de cómo puedes manipular y consultar datos utilizando LINQ en C#.

Listo. Pulsa ENTER para salir.
```

## Descripciones generadas por OpenAI

OpenAI fue utilizado para explicar automáticamente las consultas LINQ, resumir los resultados y clasificar los posts. Gracias a esto, se logró obtener interpretaciones naturales del código, generando valor agregado al análisis. Algunos ejemplos de prompts usados fueron:

- Explica brevemente en lenguaje natural qué hace esta consulta LINQ.
- Resume en 3 líneas el contenido general de estos datos.
- Clasifica los siguientes posts en temas o categorías generales.

## Conclusión

Aplicar el uso de inteligencia artificial, en conjunto con C# y LINQ, demostró ser una herramienta poderosa para el análisis de datos. Permite no solo obtener resultados inteligentes mediante consultas, sino también interpretaciones automatizadas generadas automáticamente. La integración con APIs REST facilita la obtención de datos dinámicos, mientras que OpenAI ofrece una capa de comprensión contextual que mejora la interpretación y presentación de los resultados. La combinación de estas tecnologías favorece el desarrollo de aplicaciones más inteligentes, analíticas y adaptadas al procesamiento automatizado de información.