PRÁCTICA 1

Extracción de datos masivos de internet

Israel Aznar Villegas

Contenido

Introducción

En esta práctica elegiremos un proveedor que nos proporcione una api para acceder a sus datos. Para ello, escribiremos un programa en Python que realice operaciones de extracción de datos a través de su api y almacene la información obtenida, en mi caso json.

API elegida

La api escogida es de RAWG. RAWG proporciona acceso a una de las bases de datos más grande de videojuegos del mundo. Para poder usar la api necesitaremos una API key o podrían banear las peticiones. Para ello, necesitaremos tener una cuenta que podemos crear de manera gratuita y nos darán una API key que podremos utilizar.

La api contiene diferentes tipos de datos. Estos son: creator-roles, creators, developers, games, genres, platforms, publishers, stores y tags. A continuación, enseñaré el formato de los datos de cada uno de ellos.

Creator-roles

Lista los tipos de roles que puede tener una persona en la creación de un videojuego (por ejemplo, diseñador, programador, artista, escritor, etc.).


```
{
    "count": 0,
    "next": "http://example.com",
    "previous": "http://example.com",
  - "results": [
      - {
            "id": 0,
            "name": "string",
            "slug": "string"
        }
    ]
}
```


Creators

Devuelve información sobre personas o estudios individuales que han trabajado en el desarrollo de videojuegos.


```
{
   "count": 0,
   "next": "http://example.com",
   "previous": "http://example.com",
 - "results": [
     - {
           "id": 0,
           "name": "string",
           "slug": "string",
           "image": "http://example.com",
           "image_background": "http://example.com",
           "games_count": 0
       }
   ]
}
```

Developers

Contiene datos sobre empresas o equipos responsables de desarrollar los videojuegos.

Games

Es el núcleo de la API; permite buscar, listar y obtener información detallada de videojuegos

Genres

Agrupa los videojuegos por categorías amplias de jugabilidad, como *Action*, *Adventure*, *RPG*, *Shooter*, *Puzzle*, entre otros.

Platforms

Muestra las distintas plataformas o consolas donde se pueden jugar los títulos.

Publishers

Incluye a las compañías que publican o distribuyen los videojuegos, encargadas de su comercialización.

Stores

Representa las tiendas o servicios digitales donde se pueden adquirir los juegos, como *Steam*, *Epic Games Store*, *PlayStation Store*, etc.

Tags

Ofrece etiquetas o descriptores temáticos que clasifican los juegos por características específicas, como *Horror*, *Multiplayer*, *Open World*, *Anime*, etc.

Aplicación

Rawg_get

Para poder acceder a los datos de la base de datos tenemos que usar la API. Para ello, he creado una función que actúa como cliente genérico para acceder a la API, todas las funciones la usan para acceder a los datos. Además, gestiona automáticamente la autenticación, la paginación y los errores en el límite de peticiones. En caso de recibir una respuesta HTTP 429, es decir, que haya demasiadas solicitudes, espera el tiempo necesario antes de volver a intentarlo, garantizando un uso ético y conforme a las restricciones de la API. También permite descargar varias páginas de resultados mediante un parámetro (allow_pages) acumulando todos los datos en una sola lista.

```python
def rawg_get(path, params=None, allow_pages=False, max_pages=5):
    if params is None:
        params = {}
    params['key'] = API_KEY

    results = []
    url = f"{BASE_URL}{path}"
    page_count = 0

    while url and (not allow_pages or page_count < max_pages):
        while True:
            r = requests.get(url, params=params if page_count == 0 else None)
            if r.status_code == 429:
                retry_after = 5
                try:
                    detail = r.json().get('detail', '')
                    if 'in' in detail:
                        import re
                        m = re.search(r'in (\d+) second', detail)
                        if m:
                            retry_after = int(m.group(1))
                except Exception:
                    pass
                print(f"429 detectado, esperando {retry_after}s...")
                time.sleep(retry_after)
                continue
            elif r.status_code != 200:
                raise RuntimeError(f"Error HTTP {r.status_code}: {r.text}")
            break

        data = r.json()
        if allow_pages and 'results' in data:
            results.extend(data['results'])
            url = data.get('next')
            page_count += 1
            time.sleep(REQUEST_DELAY)
        else:
            return data

    return results
```

Funciones

Todos los datos que obtengamos con las diferentes funcionas que se enseñarán a continuación su almacenan en json utilizando la siguiente función:

```python
def guardar_json(nombre_funcion, data):
    os.makedirs("json_results", exist_ok=True)
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"json_results/{nombre_funcion}_{timestamp}.json"

    if isinstance(data, pd.DataFrame):
        content = data.to_dict(orient="records")
    else:
        content = data

    with open(filename, "w", encoding="utf-8") as f:
        json.dump(content, f, ensure_ascii=False, indent=2)

    print(f"✅ JSON guardado: {filename}")
```

Juegos por fecha

Esta función obtiene los títulos y calificaciones de los videojuegos lanzados entre dos fechas determinadas. Utiliza el parámetro *dates* de la API para filtrar los resultados por rango temporal y los ordena de mayor a menor puntuación (-rating). La función recopila varias páginas de resultados gracias a la paginación (allow_pages=True), creando un conjunto de datos con el nombre del juego, su fecha de lanzamiento, su valoración media de usuarios y la puntuación de Metacritic. Finalmente, los resultados se convierten en un DataFrame de pandas y se guardan en un archivo JSON. Esta función resulta útil para analizar la evolución de la calidad de los juegos en un periodo concreto o comparar tendencias por años.

```python
def juegos_por_fecha(fecha_inicio, fecha_fin, max_pages=5):
    path = '/games'
    params = {'dates': f'{fecha_inicio},{fecha_fin}', 'ordering': '-rating', 'page_size': 40}
    juegos = rawg_get(path, params=params, allow_pages=True, max_pages=min(max_pages, MAX_PAGES))
    df = pd.DataFrame([{
        'id': j.get('id'),
        'name': j.get('name'),
        'released': j.get('released'),
        'rating': j.get('rating'),
        'metacritic': j.get('metacritic')
    } for j in juegos])
    guardar_json("juegos_titulos_calificaciones", df)
    return df
```

Juegos de un desarrollador

Esta función permite obtener todos los juegos publicados por un desarrollador específico. Primero realiza una búsqueda en el endpoint /developers de la API para identificar el ID correspondiente al nombre del estudio introducido. Luego utiliza ese identificador para solicitar la lista completa de juegos desarrollados por esa empresa. Los resultados incluyen el título, fecha de lanzamiento y valoración de cada juego. Gracias a la opción de paginación, la función puede recopilar múltiples páginas, garantizando un conjunto de datos representativo. Su principal utilidad radica en analizar la trayectoria o rendimiento de un estudio concreto dentro del mercado de los videojuegos.

```python
def juegos_de_desarrollador(nombre_dev, max_pages=10):
    # buscar desarrollador para obtener id
    data = rawg_get('/developers', params={'search': nombre_dev, 'page_size': 10})
    devs = data.get('results', []) if isinstance(data, dict) else []
    if not devs:
        return pd.DataFrame()
    dev_id = devs[0]['id']
    # luego listar juegos del desarrollador
    juegos = rawg_get(f'/games', params={'developers': dev_id, 'page_size': 40}, allow_pages=True, max_pages=min(max_pages, MAX_PAGES))
    df = pd.DataFrame([{'id': j['id'], 'name': j['name'], 'released': j.get('released'), 'rating': j.get('rating')} for j in juegos])
    guardar_json("juegos_de_desarrollador", df)
    return df
```

Top de juegos por género

Esta función consulta la API de RAWG para obtener los juegos mejor valorados de un género concreto. Primero, busca en el endpoint /genres los géneros que coincidan con el texto introducido por el usuario (nombre_genero). Si encuentra resultados, compara el nombre dado con los géneros devueltos y elige el más parecido; si no hay coincidencia cercana, toma el primero. Luego obtiene los juegos de ese género desde el endpoint /games, ordenados por puntuación (-rating), y construye un DataFrame con su nombre, calificación y fecha de lanzamiento, filtrando los que no tengan esos datos. Finalmente, guarda los resultados en un archivo JSON con guardar_json y devuelve los top_n juegos con mejores valoraciones.

```python
def top_juegos_genero(nombre_genero, top_n=20):

    data = rawg_get('/genres', params={'search': nombre_genero})
    results = data.get('results', []) if isinstance(data, dict) else []

    if not results:
        print(f"Género '{nombre_genero}' no encontrado.")
        return pd.DataFrame()

    nombres_generos = [g['name'] for g in results]
    mejor_coincidencia = difflib.get_close_matches(nombre_genero, nombres_generos, n=1, cutoff=0.4)

    if mejor_coincidencia:
        match = next(g for g in results if g['name'] == mejor_coincidencia[0])
    else:
        match = results[0]

    slug = match['slug']

    juegos = rawg_get(
        '/games',
        params={'genres': slug, 'ordering': '-rating', 'page_size': 40},
        allow_pages=True,
        max_pages=3
    )

    df = pd.DataFrame([
        {'name': j['name'], 'rating': j.get('rating'), 'released': j.get('released')}
        for j in juegos if j.get('released') and j.get('rating')
    ])

    guardar_json("top_juegos_genero", df)

    return df.head(top_n)
```

Top juego por etiqueta

Esta función obtiene los juegos mejor valorados que pertenecen a una etiqueta o "tag" específica dentro de la API de RAWG (por ejemplo, *Horror*, *Multiplayer* u *Open World*). Primero busca el tag en el endpoint /tags usando el texto proporcionado por el usuario y toma el primer resultado de la búsqueda. Luego extrae su identificador y realiza otra consulta al endpoint /games, pidiendo los juegos asociados a ese tag, ordenados por valoración. Después, construye un DataFrame con los nombres, puntuaciones y fechas de lanzamiento de los juegos, excluyendo aquellos sin datos de rating o released. Finalmente, guarda los resultados en un archivo JSON mediante guardar_json y devuelve solo los top_n juegos más destacados según la puntuación obtenida.

```python
def top_juegos_tags(nombre_tag, top_n=20):
    data = rawg_get('/tags', params={'search': nombre_tag})
    results = data.get('results', []) if isinstance(data, dict) else []

    if not results:
        print(f"Etiqueta '{nombre_tag}' no encontrada.")
        return pd.DataFrame()

    match = results[0]
    slug = match['slug']

    juegos = rawg_get(
        '/games',
        params={'tags': slug, 'ordering': '-rating', 'page_size': 40},
        allow_pages=True,
        max_pages=3
    )

    df = pd.DataFrame([
        {'name': j['name'], 'rating': j.get('rating'), 'released': j.get('released')}
        for j in juegos if j.get('released') and j.get('rating')
    ])

    guardar_json("top_juegos_tags", df)

    return df.head(top_n)
```

Próximos juegos

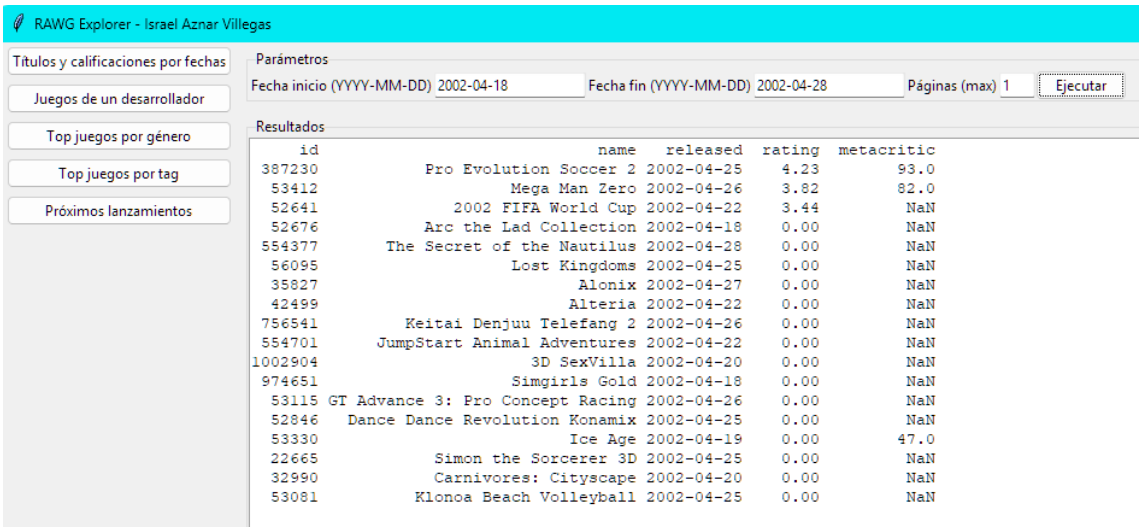
Esta función obtiene una lista de los juegos más recientes o próximos a lanzarse, tomando como referencia la fecha actual del sistema. Calcula un rango de fechas que abarca desde hoy hasta el número de meses indicados por el usuario, y consulta el endpoint /games ordenando los resultados por fecha de lanzamiento (released). El resultado es un conjunto de juegos con su nombre, fecha prevista y puntuación, útil para realizar seguimientos del mercado o identificar próximos lanzamientos de interés. Esta función destaca por combinar manejo de fechas dinámico y filtrado temporal preciso

```python
def proximos_lanzamientos(meses_hacia_adelante=3, top_n=50):
    hoy = datetime.~~utcnow~~().date()
    fin = hoy + pd.DateOffset(months=meses_hacia_adelante)
    path = '/games'
    params = {'dates': f'{hoy},{fin.date()}', 'ordering': 'released', 'page_size': 40}
    juegos = rawg_get(path, params=params, allow_pages=True, max_pages=3)
    df = pd.DataFrame([{'name': j['name'], 'released': j.get('released'), 'rating': j.get('rating')} for j in juegos])
    guardar_json("juegos_próximos", df)
    return df.head(top_n)
```

Visualización

Para poder usar la práctica lo único que hay que hacer es ejecutar el código y luego introducir los valores correspondientes según lo que necesitemos y, una vez introducidos y seleccionada la categoría, pulsar ejecutar y esperar a los resultados.

A continuación, mostraré algunas imágenes de la aplicación en funcionamiento y sus resultados. Para poder hacerlo más visual, se ha utilizado tkinter.

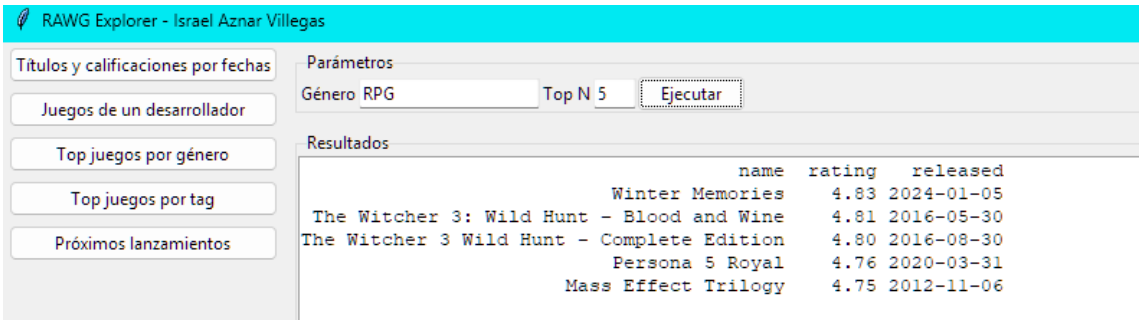


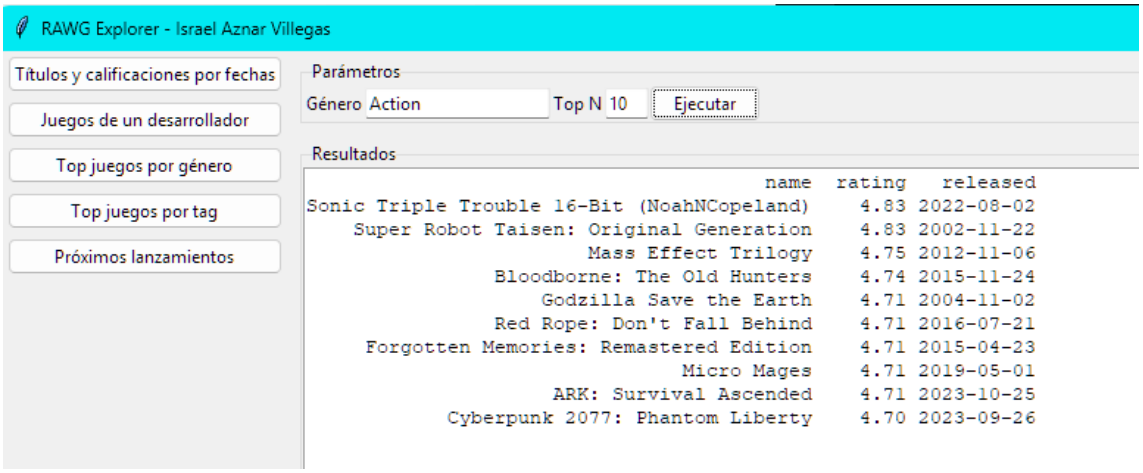
id	name	released	rating	metacritic
387230	Pro Evolution Soccer 2	2002-04-25	4.23	93.0
53412	Mega Man Zero	2002-04-26	3.82	82.0
52641	2002 FIFA World Cup	2002-04-22	3.44	NaN
52676	Arc the Lad Collection	2002-04-18	0.00	NaN
554377	The Secret of the Nautilus	2002-04-28	0.00	NaN
56095	Lost Kingdoms	2002-04-25	0.00	NaN
35827	Alonix	2002-04-27	0.00	NaN
42499	Alteria	2002-04-22	0.00	NaN
756541	Keitai Denjuu Telefang 2	2002-04-26	0.00	NaN
554701	JumpStart Animal Adventures	2002-04-22	0.00	NaN
1002904	3D SexVilla	2002-04-20	0.00	NaN
974651	Simgirls Gold	2002-04-18	0.00	NaN
53115	GT Advance 3: Pro Concept Racing	2002-04-26	0.00	NaN
52846	Dance Dance Revolution Konamix	2002-04-25	0.00	NaN
53330	Ice Age	2002-04-19	0.00	47.0
22665	Simon the Sorcerer 3D	2002-04-25	0.00	NaN
32990	Carnivores: Cityscape	2002-04-20	0.00	NaN
53081	Klonoa Beach Volleyball	2002-04-25	0.00	NaN

Podemos ver como nos devuelve todos los juegos entre dos fechas dadas. En este caso, como solo hay 10 dias no hay tantos videojuegos y no haría falta muchas páginas. Si quisiésemos un rango de fechas mas grande, deberíamos especificar las páginas que queremos porque con una no veríamos todos los resultados.

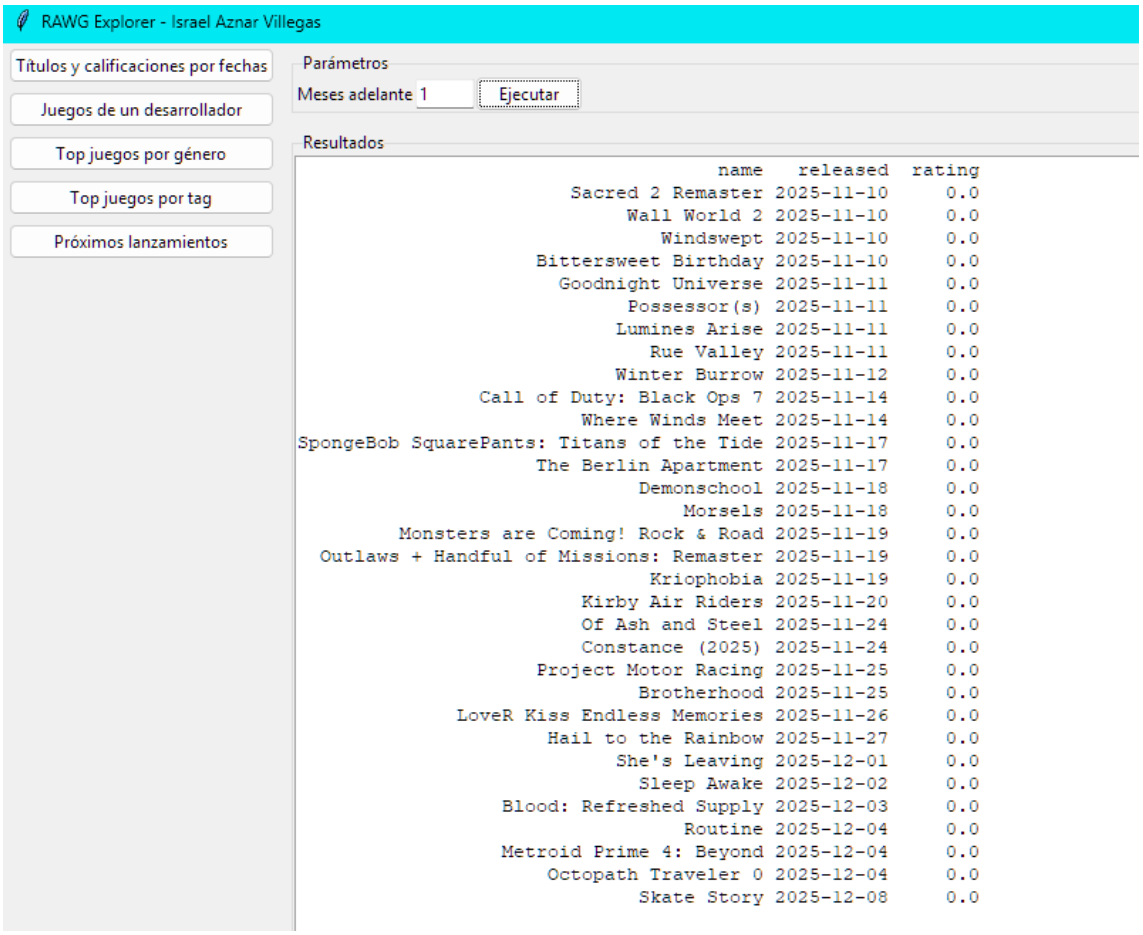
50734	Sekiro: Shadows Die Twice	2019-03-22	4.38
326243	Elden Ring	2022-02-25	4.38
3371	Dark Souls II: Scholar of the First Sin	2015-04-01	4.15
14962	Dark Souls: Prepare To Die Edition	2012-08-23	4.37
51610	Dark Souls: Remastered	2018-05-23	4.47
5538	Dark Souls	2011-09-22	4.34
3751	Dark Souls II	2014-03-11	4.03
4108	Demon's Souls	2009-02-05	4.28
892902	Armored Core VI: Fires of Rubicon	2023-08-25	4.12
12988	Ninja Blade	2009-01-29	3.36
977470	Elden Ring: Shadow of the Erdtree	2024-06-21	4.52
43737	Dark Souls III: Ashes of Ariandel	2016-10-25	4.46
385406	Dark Souls III: The Ringed City	2017-03-28	4.61
28764	Enchanted Arms	2006-01-12	3.29
58791	Metal Wolf Chaos XD	2019-08-05	3.00
28889	Chromehounds	2010-01-07	3.27
605544	Bloodborne: The Old Hunters	2015-11-24	4.74
997548	Elden Ring Nightreign	2025-05-30	3.87
836449	Bloodborne: Game of the Year Edition	2015-11-27	4.24
4245	ARMORED CORE V	2012-01-26	3.29
42767	Dark Souls II: Crown of the Sunken King	2014-07-22	3.97
510042	Dark Souls: Artorias of the Abyss	2012-08-24	4.35
42848	Dark Souls II: Crown of the Ivory King	2014-09-30	3.97
281852	Kuon	2004-04-01	3.67
39454	Armored Core 4	2006-12-21	3.57
42847	Dark Souls II: Crown of the Old Iron King	2014-08-26	4.00
28808	AC for Answer	2010-04-10	0.00
52681	Armored Core	1997-07-10	4.13
244718	Dark Souls Trilogy	2019-03-01	4.09
3901	Armored Core: Verdict Day	2013-09-24	0.00
407549	Tenchu: Fatal Shadows	2004-07-22	4.20
52899	Echo Night	1999-07-31	3.91
53075	King's Field	1994-12-16	3.09
57755	Shadow Tower (1999)	1999-10-31	2.89
53078	King's Field II	1995-07-21	3.33
362612	Echo Night: Beyond	2004-07-27	0.00
39455	Armored Core: For Answer	2008-03-19	0.00

Si la empresa que pongamos tiene pocos juegos no necesitaremos paginar que nos devuelven todos y no tardaría mucho. Sin embargo, si ponemos por ejemplo Nintendo, deberíamos paginar ya que existen demasiados juegos.





En estos casos muy similares podemos ver cómo nos devuelve los n mejores según nuestro interés.



Esta función es muy interesante por si estamos interesados en algún juego futuro.

Conclusión

Con esta práctica he podido afianzar mis conocimientos sobre el uso de APIs y aprender a plantear estrategias para resolver problemas comunes, como la paginación o el error 429, implementando esperas entre peticiones. Además, me ha ayudado a comprender mejor cómo estructurar consultas y procesar la información obtenida de forma eficiente.