



Web Scrapping en Fuentes de Datos

03/01/2026

Christian Edgardo Carrillo Aburto

Prueba Técnica (Ejercicio 1)

Introducción

El presente documento describe el desarrollo de un sistema basado en un REST API que emplea técnicas de web scraping para consultar fuentes públicas de alto riesgo, devolver los resultados de forma estructurada y facilitar su revisión por parte de analistas de riesgo

Descripción del Sistema

Objetivo General

Desarrollar un REST API que permita realizar búsquedas automatizadas de entidades en listas públicas de alto riesgo mediante técnicas de web scraping, retornando los resultados de manera estructurada y legible para su análisis.

Objetivo Específico

1. Automatizar la búsqueda de entidades en fuentes públicas de alto riesgo
2. Extraer información relevante utilizando técnicas de web scraping.
3. Retornar el número de coincidencias encontradas para cada búsqueda.
4. Proveer un servicio consumible mediante una colección de Postman.
5. Implementar mecanismos básicos de autenticación y control de uso del API.

Alcance del Sistema

Incluye:

- Búsqueda de entidades por nombre
- Consulta en al menos una fuente pública de alto riesgo
- Retorno de resultados en formato JSON
- Implementación de autenticación mediante API Key
- Control de número máximo de solicitudes por minuto.

Usuarios del Sistema

- Analistas de riesgo
- Sistemas automatizados que consuman el API.

Requerimientos del Sistema

Requerimientos Funcionales

ID	Descripción
RF-01 Búsqueda de entidades	El sistema debe permitir realizar búsquedas de entidades ingresando su nombre a través de un endpoint REST
RF-02 Uso de web scraping	El sistema debe obtener la información mediante técnicas de web scraping sobre al menos una fuente pública de alto riesgo
RF-03 Retorno de resultados	El sistema debe retornar el número de coincidencias encontradas y un arreglo con los resultados obtenidos.
RF-04 Formato de respuesta	El sistema debe devolver los resultados en formato JSON.
RF-05 Manejo de errores	El sistema debe mostrar mensajes de error claros ante entradas inválidas o fallos en la consulta.

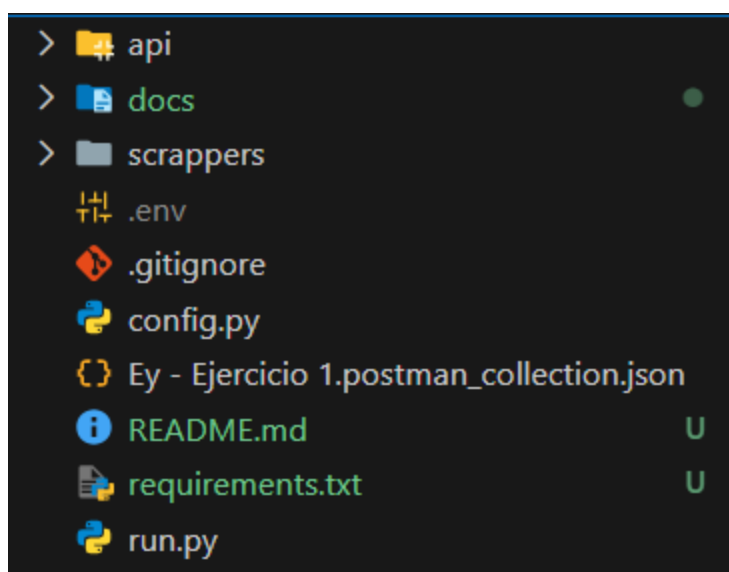
Requerimientos No Funcionales

ID	Descripción
RNF-01 Autenticación	El sistema debe requerir autenticación mediante API Key para el consumo del servicio.
RNF-02 Límite de llamadas	El sistema debe permitir un máximo de 20 solicitudes por minuto por cliente

Arquitectura del Sistema

Este sistema está construido siguiendo una arquitectura por capas que separa claramente las responsabilidades y facilita el mantenimiento del código. El sistema utiliza FastAPI como framework principal, aprovechando sus capacidades asíncronas para ejecutar múltiples operaciones de scraping en paralelo, reduciendo significativamente los tiempos de respuesta.

La arquitectura está diseñada para ser resiliente ante fallos individuales de fuentes externas, permitiendo que si una fuente falla (por ejemplo, OFAC), las demás continúen funcionando normalmente. Esto garantiza que el usuario siempre reciba la máxima información posible disponible en cada momento.




Componentes Principales

El sistema se compone de cuatro capas principales que trabajan en conjunto:

1. Capa de API (API Layer)

Esta es la capa de entrada del sistema donde se exponen los endpoints REST. Utiliza FastAPI para definir rutas, validar datos de entrada mediante Pydantic, y gestionar las respuestas en formato JSON. Cada endpoint está documentado automáticamente



mediante OpenAPI/Swagger, permitiendo una exploración interactiva de la API. Los middlewares interceptan las peticiones antes de llegar a los controladores, inyectando funcionalidad transversal como headers de rate limiting.

2. Capa de Seguridad (Security Layer)

Antes de procesar cualquier petición, el sistema valida la autenticidad del usuario mediante API Keys. Las claves se almacenan hasheadas con SHA256 en variables de entorno, nunca en texto plano. Adicionalmente, el rate limiter controla que ningún usuario supere las 20 peticiones por minuto, protegiendo tanto al sistema como a las fuentes externas de un uso excesivo. Esta capa también agrega headers informativos en cada respuesta indicando al cliente cuántas peticiones le quedan disponibles.

3. Capa de Lógica de Negocio (Business Logic)

Aquí se orquesta la ejecución de las búsquedas. Cuando un usuario solicita una búsqueda multi-fuente (/api/v1/search/all), la cual se encarga de hacer una búsqueda en las 3 fuentes provistas. La capa consolida los resultados, calcula totales, y formatea la respuesta final.

4. Capa de Scrapers (Data Collection)

Cada fuente externa tiene su propio scraper especializado que conoce la estructura particular de ese sitio web o API:

OFAC Scraper: Utiliza Playwright para interactuar con el formulario de búsqueda del sitio del Tesoro de EE.UU. Simula comportamiento humano con delays aleatorios para evitar bloqueos.

Offshore Leaks Scraper: Es el más complejo debido a la protección AWS WAF del sitio ICIJ. Implementa técnicas avanzadas como scrolling progresivo, delays aleatorios entre 4-10 segundos, y detección de challenges de verificación humana. Si detecta un CAPTCHA, detiene gracefully y retorna los resultados parciales obtenidos.

World Bank Scraper: Consume directamente la API REST del Banco Mundial, lo que lo hace el más rápido y confiable. Realiza una única llamada para obtener todas las empresas sancionadas y luego filtra localmente por nombre.

Diseño del REST API

Endpoint Principal

GET `/api/v1/search/all`

Payload de Entrada

```
{  
  "entity_name": "MAX"  
}
```

Respuesta de la API

```
{  
  "hits": 3,  
  "results": [  
    {  
      "source": "Offshore Leaks Database",  
      "matches": [  
        {  
          "entity": "MAXIMUS INC",  
          "jurisdiction": "US",  
          "linked_to": "John Doe",  
          "data_from": "2020-01-15"  
        }  
      ]  
    },  
    {  
      "source": "OFAC Sanctions Search",
```

```
"matches": [  
  {  
    "name": "MAXWELL LTD",  
    "address": "123 Main St, Anytown, USA",  
    "type": "Individual",  
    "programs": ["SDN"],  
    "list": "Specially Designated Nationals",  
    "score": 95  
  },  
  {  
    "name": "MAX CORPORATION",  
    "address": "456 Elm St, Othertown, USA",  
    "type": "Entity",  
    "programs": ["SSI"],  
    "list": "Sectoral Sanctions Identifications",  
    "score": 88  
  }  
]  
}
```

Manual de Usuario (Postman)

Para ejecutar el sistema, el usuario deberá importar la colección de Postman proporcionada junto con la solución en el repositorio.

Una vez importada, podrá configurar los encabezados de autenticación y ejecutar las solicitudes disponibles para realizar búsquedas de entidades en listas de alto riesgo.

Ey - Ejercicio 1

Colección de API's para el Ejercicio de la prueba técnica de EY.

Error Cases

Una serie de request en las que observaremos distintos comportamientos de los endpoints.

GET No API KEY [Open request→](#)

```
/api/v1/rate-limit
```

JUMP TO

- Introduction
- > [Error Cases](#)
- [GET Health](#)
- [POST Search All Sources](#)
- [POST Scrapping Ofac](#)
- [POST Scrapping Offshorelake](#)
- [POST Scrapping World Bank](#)
- [GET Rate Limit](#)

Manual Técnico

Requisitos Previos

Software requerido:

- Python 3.9 o superior : Entorno de ejecución principal
- pip 21.0+: Gestor de paquetes de Python
- Git: Para clonar el repositorio del proyecto.
- Chromiun / Chrome: Utilizado por Playwright para web scraping.

Despliegue

1. Clonar el repositorio
git clone <https://github.com/StylezZz/Ey-Prueba-Ejercicio-1.git>
cd Ey-Prueba-Ejercicio 1
2. Instalar dependencias
pip install -r requirements.txt
playwright install chromium
3. Configurar variables de entorno
Crear un archivo .env en la raíz del proyecto

4. Ejecutar la aplicación
python [run.py](#)

5. Verificar despliegue
Health check

curl <http://localhost:8000/health>

Manejo de Errores y Validaciones

Validaciones de Entrada

Validación de API Key:

Error 401 - No API Key

```
{  
  "error": "API Key is required. Please provide a valid API key in the X-API-Key header.",  
  "detail": "API Key is required. Please provide a valid API key in the X-API-Key header.",  
  "timestamp": "2026-01-03T00:47:30.744124"  
}
```

Error 403 - API Key inválida

```
{  
  "error": "Invalid API Key. Please check your credentials.",  
  "detail": "Invalid API Key. Please check your credentials.",  
  "timestamp": "2026-01-03T00:51:12.577919"  
}
```

Rate Limiting

Límite: 20 request por minuto

Error 429 - Rate limit excedido

```
{  
  "error": "Rate limit exceeded",  
  "detail": "Maximum 20 requests per 60 seconds allowed",  
  "timestamp": "2026-01-03T00:55:46.156689"  
}
```

Pruebas Realizadas

Herramientas utilizadas

- Postman: Pruebas funcionales de endpoints
- Collection incluida: Ey-Ejercicio 1.postman_collection.json

Casos de prueba

Test 1: Health Check

GET /health

Status: 200 OK

Test 2: Búsqueda exitosa en OFAC

POST /api/v1/search/ofac

Body: {"entity_name": "Miami"}

Status: 200 OK

Test 3: Búsqueda en todos las fuentes

POST /api/v1/search/all

Body: {"entity_name": "Miami"}

Status: 200 OK

Limitaciones y Mejoras Futuras

Limitaciones actuales

Dependencia de estructura HTML:

- **Problema:** Cambios en las páginas web pueden romper los scrapers
- **Fuentes afectadas:** OFAC y Offshore Leaks
- **Mitigación actual:** logs detallados y manejo de excepciones.

Challenge humano en Offshore Leaks:

- **Problema:** Detección de bot activa verificación CAPTCHA
- **Impacto:** Muy Alto, ya que puede bloquear el scraping y no encontrar resultados.
- **Mitigación:** Delays aleatorios y simulaciones de scroll humano.

Mejoras futuras propuestas

- Implementar caché Redis, para poder reducir el tiempo de respuesta a <1s para las búsquedas repetitivas
- Una base de datos para poder almacenar historial de búsquedas.
- Añadir más fuentes de datos.