

# Desafío Técnico

## Consigna

El desafío técnico consiste en armar un servicio web que exponga un endpoint para leer un archivo, consultar una serie de **APIs de Mercado Libre** y cargar una base de datos con los datos del archivo y las consultas a las APIs.

- Microservicio
  - Se debe levantar usando Gin(Go) o Flask(Python).
  - Debe exponer un endpoint que lance el ejercicio requerido. Puede haber más si se desea, pero no es necesario.
- Lectura del archivo a cargar
  - El archivo consta de 2 columnas
    - site (string): país de donde es el ítem (Argentina, Brasil). Cada site tiene un código; Argentina es MLA, Brasil es MLB, etc.
    - id (numérico): código de identificación del ítem. La clave de un ítem consta del site y el id concatenado. Si el site es “FOO” y el id es “123”, la clave de un ítem es “FOO123”.
  - La lectura del archivo se debe poder configurar, y debe ser parte del programa (formato, separador, encoding, etc). Si se cambiara algo del archivo (por ejemplo, separador o formato), debe poder leerse cambiando la configuración.
  - Debe soportar formatos streamables (ej.: csv, jsonlines, txt). Formatos como JSON que no lo son, pueden no ser soportados; no es parte del ejercicio.
  - En el ejemplo son pocas líneas. Trabajarlo como si fuera un archivo más grande que la memoria.
- Consulta a las APIs
  - Debe hacerse con la premisa de la lectura del archivo: realizar las consultas de la manera más rápida posible, de la manera más eficiente en cuanto a uso de recursos del sistema operativo.
  - Dado que la **API de ítems** de Mercado Libre **no es pública**, genera mocks basada en la documentación de [API Docs](#), es permitido el uso de AI para esto:
    - Con la data del archivo, consulta la API de Ítems generada por ti. A partir de la respuesta:
      - Quedarse con el campo `price`
      - Con el `category\_id` pegarle a la api de categorías y traerse el campo `name`
      - Con el `currency\_id` pegarle a la api de currencies y traerse el campo `description`
- Base de datos
  - Los campos a insertar finalmente van a ser:
    - site (obtenido del file)
    - id (obtenido del file)
    - price (obtenido de la api de ítems)
    - name (obtenido de la api de categorías)
    - description (obtenido de la api de monedas)
  - Se debe levantar en un container. El motor puede ser no relacional, está a criterio del desarrollador.
  - No es necesario escribir el código del container. Se puede utilizar una imagen existente. A criterio del desarrollador.

- Se debe poder acceder a la tabla cargada con los datos.
- Documentación
  - Se debe entregar la documentación necesaria para ejecutar el *challenge* de manera fluida y entender el proyecto. Cuánta documentación es necesaria queda a criterio del desarrollador.
- Testing
  - No son necesarios para el desafío. Si se conoce un framework de testing, implementar tests unitarios simples para demostrar el nivel de conocimiento del framework.

## ¿Qué se va a evaluar?

- En general
  - Uso de módulos y librerías: evitar usar librerías cuando un módulo built-in lo puede resolver.
  - Buenas prácticas de código y patrones de diseño.
- Microservicio
  - Estructura del proyecto. Flexibilidad (“*que los cambios no duelan*”) y modularidad (separación de responsabilidades).
- Lectura del archivo
  - Construcción de buenas abstracciones para que sea fácil soportar múltiples formatos a través de configuración.
  - Evitar *spaghetti code*
- Consulta a las APIs
  - Selección de una manera eficiente en cuanto a recursos y rápida de interactuar con las APIs.
- Base de datos
  - Elección del motor. Hay múltiples opciones y ninguna es necesariamente mejor que otra. Elegir la que más cómoda.
- Documentación
  - Arquitectura de la aplicación.
  - Claridad suficiente para poder ejecutar la solución punta a punta, desde el build del container hasta la carga de la base de datos.
  - La buena documentación es importante para el mantenimiento y el trabajo en equipo, así que es algo a lo que le vamos a dar importancia.
- Testing
  - Opcional, pero suma puntos :)

# Desafío Teórico

## Consigna

### Procesos, hilos y corrutinas

- Un caso en el que usarías procesos para resolver un problema y por qué.
- Un caso en el que usarías threads para resolver un problema y por qué.
- Un caso en el que usarías corrutinas para resolver un problema y por qué.

### Optimización de recursos del sistema operativo

- Si tuvieras 1.000.000 de elementos y tuvieras que consultar para cada uno de ellos información en una API HTTP. ¿Cómo lo harías? Explicar.

### Análisis de complejidad

- Dados 4 algoritmos **A**, **B**, **C** y **D** que cumplen la misma funcionalidad, con complejidades  $O(n^2)$ ,  $O(n^3)$ ,  $O(2^n)$  y  $O(n \log n)$ , respectivamente, ¿Cuál de los algoritmos favorecerías y cuál descartarías en principio? Explicar por qué.
- Asume que dispones de dos bases de datos para utilizar en diferentes problemas a resolver. La primera llamada **AlfaDB** tiene una complejidad de  $O(1)$  en consulta y  $O(n^2)$  en escritura. La segunda llamada **BetaDB** que tiene una complejidad de  $O(\log n)$  tanto para consulta, como para escritura. ¿Describe en forma sucinta, qué casos de uso podrías atacar con cada una?

## Condiciones de Entrega

**Formato de entrega:** repositorio de Git. Se pedirá que agreguen a quien revise el ejercicio como colaborador del repositorio al entregar el ejercicio. El ejercicio teórico debe estar contestado en un README en el repositorio, en una carpeta específica.