



<b>Desafío Técnico</b>	<b>2</b>
Consigna	2
¿Qué se va a evaluar?	4
<b>Desafío Teórico</b>	<b>5</b>
Consigna	5
Procesos, hilos y corrutinas	5
Optimización de recursos del sistema operativo	5
<b>Condiciones de Entrega</b>	<b>6</b>

# 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 públicas de MercadoLibre y cargar una base de datos con los datos del archivo y las consultas a las APIs.

- Microservicio
  - Se debe levantar usando Flask.
  - 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
    - i. site (string): país de donde es el ítem (Argentina, Brasil). Cada site tiene un código; Argentina es MLA, Brasil es MLB, etc.
    - ii. 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 streameables (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.
  - Desarrollar suponiendo que se podrían agregar nuevas APIs para consultar.
  - Flujo de consumo de APIs (todas las APIs son públicas de MercadoLibre, documentación en [API Docs](#)):
    - i. Con la data del archivo, consultar la API de Ítems de MercadoLibre. A partir de la respuesta:
      - Quedarse con el campo `price`
      - Quedarse con el campo `start\_time`

- 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`
  - Con el `seller\_id` pegarle a la api de users y traerse el campo `nickname`
- Base de datos
  - Los campos a insertar finalmente van a ser:
    - i. site (obtenido del file)
    - ii. id (obtenido del file)
    - iii. price (obtenido de la api de ítems)
    - iv. start\_time (obtenido de la api de ítems)
    - v. name (obtenido de la api de categorías)
    - vi. description (obtenido de la api de monedas)
    - vii. nickname (obtenido de la api de users)
  - Se debe levantar en un container. El motor debe ser relacional, y es 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
  - 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.

# 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.