# Challenge - QA Automation

## Objetivo:

Evaluar las habilidades de un QA Automation en el diseño, implementación y automatización de pruebas para un servicio API, enfocándose en la cobertura funcional y de rendimiento, además del uso de buenas prácticas de automatización.

## Ejercicio:

1. **Diseñar pruebas funcionales con Postman para el endpoint "API – Ingreso Pedido Lista"**:
    - Crear al menos 5 casos de prueba que verifiquen las principales funcionalidades del endpoint, como:
        - Validación de un pedido exitoso.
        - Manejo de errores en campos obligatorios y/o datos incorrectos.
        - Verificación de las respuestas correctas del servidor (status codes y payload).
    - Incluir variaciones de datos de entrada para cubrir los escenarios principales.
    - Documentar los casos de prueba en Postman Collection y agregar descripciones claras para cada uno.
2. **Utilizar JMeter para diseñar un plan de pruebas de rendimiento para el método**:
    - Definir escenarios de carga que incluyan:
        - Pruebas de estrés (identificar el umbral de fallo del servicio).
        - Pruebas de carga (evaluar el rendimiento bajo carga moderada).
        - Pruebas de volumen (simular grandes cantidades de datos de entrada).
    - Generar reportes detallados con métricas clave como tiempo de respuesta, tasa de errores, throughput, etc.
    - Explicar los criterios de éxito/fallo en las pruebas de rendimiento y anexar el archivo `.jmx` utilizado.
3. **Automatizar los casos básicos de la API con Python y la librería Requests**:
    - Implementar una suite de pruebas automatizadas que cubra los siguientes casos:
        - Ingreso de un pedido exitoso.
        - Validación de campos obligatorios faltantes o incorrectos.
        - Mockear el servicio para simular respuestas utilizando un JSON de ejemplo (puedes usar librerías como `unittest.mock` o `responses`).
    - Asegurar que el código siga las buenas prácticas:
        - Organizar el código en funciones y módulos reutilizables.
        - Manejo de excepciones adecuado.

        - Crear un archivo de configuración para parámetros como la URL base y credenciales de acceso.
    - Mostrar los resultados de las pruebas en un reporte Allure para facilitar la visualización de las métricas y el estado de las pruebas.
4. **Compartir un repositorio Git con todo el trabajo realizado**:
    - Incluir en el repositorio:
        - Los archivos de Postman, JMeter y Python.
        - Un archivo `README.md` que explique:
            - Cómo ejecutar las pruebas funcionales en Postman.
            - Cómo ejecutar el plan de pruebas de JMeter.
            - Instrucciones para correr la suite de pruebas automatizadas con Python.
        - El reporte Allure generado por las pruebas automatizadas.
        - Notas sobre cualquier decisión técnica importante que se haya tomado durante el proceso.

# Buenas Prácticas:

A lo largo del challenge, asegúrate de aplicar buenas prácticas de testing y automatización:

- Nombres de variables y funciones claros y descriptivos.
- Uso de assertions específicas y detalladas.
- Mantener el código modular y fácil de mantener.
- Versionar correctamente el trabajo en el repositorio Git, con commits claros.

# Requerimiento Funcional

## API - Ingreso Pedido Lista

## Introducción

El presente documento describe el diseño de un método unificado para el ingreso de pedidos de mercadería, abarcando tanto las áreas de **Compras** como de **Ventas** en una única solicitud (request). El objetivo es permitir el procesamiento masivo de pedidos, optimizando la carga y gestión de grandes volúmenes de datos.

Este servicio será utilizado por **500 sucursales** y se estima que manejará una carga de aproximadamente **1 millón de registros semanales**.

## Detalle del Funcionamiento

El servicio permitirá procesar operaciones de **inserción** y **eliminación** (insert/delete) de múltiples registros de pedidos, tanto de **Compras** como de **Ventas**, en un solo request. Los pedidos serán almacenados temporalmente en un **buffer** o **tabla interna** en la base de datos, y

se procesarán de manera individual, lo que optimiza la gestión de grandes volúmenes de información.

## Requerimientos del Servicio

1. **Procesamiento Masivo**:
    - El servicio debe ser capaz de recibir y procesar grandes volúmenes de pedidos en una única solicitud.
    - Los registros serán procesados de manera secuencial desde el buffer hacia la tabla definitiva, permitiendo un manejo eficiente del sistema.
2. **Respuesta del Servicio**:
    - En la respuesta de cada request, el servicio deberá devolver un **ID de proceso**, el cual servirá para realizar consultas posteriores sobre el estado del procesamiento de los pedidos.
3. **Ejecución Simultánea**:
    - El sistema debe soportar la **ejecución simultánea** de múltiples listas de pedidos, garantizando que diferentes sucursales puedan enviar sus solicitudes sin interferencias entre procesos.

## Consideraciones Adicionales

- **Escalabilidad**: Dado el alto volumen de datos esperado, el servicio debe estar preparado para escalar en función de la carga, asegurando que pueda manejar el crecimiento en la cantidad de registros procesados semanalmente.
- **Concurrencia**: El diseño debe contemplar la ejecución concurrente de varios procesos de ingreso, sin que haya bloqueos o cuellos de botella en el sistema.
- **Integridad de los Datos**: Asegurarse de que el procesamiento individual de cada registro garantice la integridad de los datos, manejando adecuadamente posibles errores y devolviendo información clara y precisa.

## InsertPedidosList

El método debe permitir el envió por lista de registros

Se ubica en una nueva sección NEGOCIO
(http://localhost:6003/api/negocio/v10/insert-pedidos-list)

Marcar en el swagger los campos requeridos y opcionales

**Request**

| **Campos api fondos** | **Requerido** | **Tipo Campo** | **Descripción** |
|---|---|---|---|

| | | | |
|---|---|---|---|
| numCliente | SI | entero | Numero comitente |
| codProducto | SI | string | Código de compra del producto |
| codTipoSolicitud | SI | string | Codigo tipo operacion, solo V si es compra y C si es Venta |
| fechaPedido | NO | date | fecha de ingreso del Pedido, si no se envía tomar la del día. |
| importe | SI | decimal | Importe solicitud en moneda solicitud. Se controlara como no requerido cuando esTotal= True |
| esTotal | SI | booleano | Si abono el total de la mercadería |
| idFormaPago | SI | string | codigo interfaz forma de pago |
| idPedido | SI | string | Codigo interno para seguimiento |
| requiereAutorizacion | SI | Booleano | Parametro que define si la solicitud debe ser autorizada. |
| origen | NO | string | Fuente de ingreso de la operacion, por defecto WEB |

Json de ejemplo Solicitud de Compra


```
[
  {
```

```
    "codProducto ": 439,
    "codTipoSolicitud ": "RE",
    "fechaPedido ": "2024-09-16",
    "importe": 700000,
    "esTotal": false,
    "idFormaPago": "CB",
    "idPedido ": "H7",
    "requiereAutorizacion": false,
    "origen": "OT"
  } ]
```


**Response**

Se devuelven los siguientes campos

| **Campos api fondos** | **Tipo Campo** | **Descripción** |
|---|---|---|
| idProceso | entero | Código interno del proceso. Representa a toda la lista de pedidos procesadas. Internamente se deberá guardar la fecha del proceso. Debe ser un GUID o ID autogenerado por SQL. si rta es 200 → id>0 |
| exception | string | Informa el error en el procesamiento del lote. Si no hay error esta en nulo. si rta no es 200 → se informa este campo |


```
{
  "idProceso": 0,
  "cantidadErrores": 0,
  "cantidadOperacionesProcesadas": 0
}
```


Con el idProceso se podrá posteriormente consultar el estado o lista de los pedidos ingresados.