METROLLANTAS C. A
Gerencia de Sistemas y Tecnología
Aprovval Testing

INFORME DE PRUEBAS DDOSIFY
Api COPILOT

https://iron-staging-api.maxcodex.com/api/

INTRODUCCION

Realizado Por:	Revisado Por:	Aprobador Por:	Versión
Erik Zerpa	Xavier Núñez	Hernán Bastidas	1.0.0

INTRODUCCIÓN

Este documento se realiza con la finalidad de verificar el estatus de los servicios del api del proyecto IRON y a su vez identificar posibles casos de mejora y/o sugerencias.

ESCENARIOS DE PRUEBA

El escenario de pruebas se va a dividir en 04 bloques o segmentos, los cuales consistirán en el tiempo de 60 segundas, realizar peticiones recurrentes en bloques de 50 hasta llegar a 200, tomando este ultimo parámetro, como la cifra de transacciones que pueden ser procesadas por la cola del api.

El url a verificar será el api de sincronización de datos, ya que es el servicio que va a consumir más recursos y memoria del servidor.

Url: **URL_API**/sync_odoo?procesado_iron=1&page=10.5

1. **50 requests (60 segundos).**

```
RESULT
------------------------------------
Success Count:    50    (100%)
Failed Count:     0     (0%)

Durations (Avg):
  DNS                 :0.0079s
  Connection          :0.0002s
  TLS                 :0.0121s
  Request Write       :0.0001s
  Server Processing   :2.0041s
  Response Read       :0.0001s
  Total               :2.0244s

Status Code (Message) :Count
  200 (OK)    :50
```

2. **100 requests (60 segundos).**

Realizado Por:	Revisado Por:	Aprobador Por:	Versión
Erik Zerpa	Xavier Núñez	Hernán Bastidas	1.0.0

```
RESULT
------------------------------------
Success Count:    68    (68%)
Failed Count:     32    (32%)

Durations (Avg):
  DNS                 :0.0000s
  Connection          :0.0000s
  TLS                 :0.0000s
  Request Write       :0.0000s
  Server Processing   :0.0071s
  Response Read       :0.0000s
  Total               :2.6340s

Status Code (Message) :Count
  200 (OK)           :28
  404 (Not Found)    :40

Server Error Distribution (Count:Reason):
  32     :connection timeout
```

3. 150 requests (60 segundos).

```
RESULT
------------------------------------
Success Count:    82    (54%)
Failed Count:     68    (46%)

Durations (Avg):
  DNS                 :0.0000s
  Connection          :0.0000s
  TLS                 :0.0000s
  Request Write       :0.0000s
  Server Processing   :0.0033s
  Response Read       :0.0000s
  Total               :2.4136s

Status Code (Message) :Count
  200 (OK)           :13
  404 (Not Found)    :69

Server Error Distribution (Count:Reason):
  68     :connection timeout
```

4. **200 requests (60 segundos).**

```
✅ Successful Run: 98      49%      ❌ Failed Run: 102      51%
✅ Successful Run: 98      49%      ❌ Failed Run: 102      51%


RESULT
---------------------------------------
Success Count:     98     (49%)
Failed Count:      102    (51%)

Durations (Avg):
  DNS                   :0.0000s
  Connection            :0.0000s
  TLS                   :0.0000s
  Request Write         :0.0000s
  Server Processing     :0.0033s
  Response Read         :0.0000s
  Total                 :2.2861s

Status Code (Message) :Count
  200 (OK)             :7
  404 (Not Found)      :91

Server Error Distribution (Count:Reason):
  102      :connection timeout
```

CONCLUSIONES Y/O RECOMENDACIONES

- En archivo previo se enviaron las recomendaciones y/o sugerencias para los servicios de auth y users.
- Se tomo como muestra el servicio de sincronización de datos por el hecho de que será el api que vas será requerida y la que más recibirá peticiones fuertes.
- Aproximadamente 1 solicitud, dependiendo de la data (en este caso se cargaron 3 servicios nuevos que sincronizar desde ODOO), se demoro entre 1.5 y 2.8 segundos por responder.
- Para la primera prueba de carga se realizaron 50 requests en 60 segundos y todas las solicitudes respondieron bien.
- Para la segunda prueba de carga se realizaron 100 requests en 60 segundos y el porcentaje de éxito fue de 68%, el resto de las transacciones retornaron error 500 o time out.
- Para la tercera prueba de carga se realizaron 150 requests en 60 segundos y el porcentaje de éxito fue de 54%, el resto de las transacciones retornaron error 500 o time out.
- Para la cuarta y última prueba se realizaron 200 requests en 60 segundos y el porcentaje de éxito fue de 49%, el resto de las transacciones retornaron error 400, 500 y time out.

- Se debe revisar el proceso de carga y sincronización en el servicio, ya que según las pruebas realizadas se evidencia que el servicio a partir de 1 solicitud por segundo empieza a fallar en las respuestas.
- Se debe verificar el potencial del servidor en donde este el api, ya que, si esto será un proceso automático generado desde la infraestructura, va a generar que mucha información no se sincronice de forma correcta.
- Verificar si por la paginación se puede extraer un mejor rendimiento para las respuestas del api.
- Según lo que se evidencia en la paginación del servicio, el de igual forma cuenta el total de ítems a sincronizar por página, y esta consulta posiblemente sea bastante pesada, para ello delimitado por el count por página, solo contar el total por página actual.
- Si se necesita el dato para saber el total de data a sincronizar, sería más optimo dejarlo en un endpoint aparte.
- La muestra máxima fue de 200 requests porque el proyecto tiene programado procesar lotes de 200 transacciones por hilo de carga.

- Se recomienda el uso de **git**, y manejar las ramas muy ordenadamente dependiendo del identificador del ticket en Jira, una propuesta de organización podría ser de la siguiente manera:
    - Rama **main**: Dedicada al proyecto en producción, y con accesos limitados.

- Rama **develop**: Una vez que se finalicen los desarrollos, estos una vez probados por el desarrollador, deben ser mergeados a la rama develop a fin de ser testeados por el equipo de pruebas.
- Rama **feature/ID_INCIDENCIA**: Se deben crear ramas identificando que son feature cuando son desarrollos o funcionalidades nuevas.
- Rama **fix/ID_INCIDENCIA**: Se deben crear ramas identificando que son fix, cuando son correcciones pequeñas o rápidas y no necesariamente dañan algún proceso.
- Rama **bug/ID_INCIDENCIA**: Se deben crear ramas identificando que son bug, cuando son errores del servicio y esto ocasiona que no se pueda seguir el proceso.
- Adicionalmente, todos los merges a las ramas develop y/o main deben realizarse a través de merge requests de gitab, y que sea el supervisor inmediato que verifique lo que se esta subiendo y proceda a aprobar o rechazar la unión de código.