



# CONEXIÓN API - DOCUMENTACIÓN

Datamining labs 2023

## ▼ Cronograma:

Frecuencia: actualización semanal

- El proceso de actualización de base BIT de pre-aprobados corre todos los miércoles a las 19:00 hrs.
- El script de Python corre automáticamente todos los jueves a las 7:00 hrs, tomando la base bit de pre-aprobados y cargandola en la pestaña de Google sheets
  - → BIT PRE-APROBADOS
- El script de R corre todos los jueves a las 11:00 hrs donde notifica el estado del proceso de python, en caso de falla adjunta el archivo para que se realice su carga manualmente (*Por el momento desde la Notebook de JMP, próximamente desde la VDI*)

## ▼ Diagrama:

```
https://prod-files-secure.s3.us-west-2.amazonaws.com/23664270-d02  
3-4aa0-a777-74d926804f90/b4c435e0-b448-4e70-9c44-66efec586d  
b8/API_SHEETS.pdf
```

## ▼ Código:

### ▼ PRODUCTIVO

```
#####  
# Importo las libr  
#####  
  
import requests  
import pandas as pd  
import json  
import logging  
from datetime import datetime  
import urllib3  
urllib3.disable_warnings(urllib3.exceptions.InsecureReques  
  
#####  
# Cargo el token y  
#####  
API_ENDPOINT = "https://sheetdb.io/api/v1/"  
API_TOKEN = ""  
SHEET_NAME = ""  
  
#####  
#PASO 1 -> REALI  
#####  
# Make a GET requ
```

```

response = requests.get(API_ENDPOINT + SHEET_NAME, headers
data = response.json()
df = pd.DataFrame(data=data)

#####
#PASO 2 -> GENER
#####
#GUARDAR EXCEL

out_path = "L:/02-Tareas_programadas/Juanma/21-API/back_up
writer = pd.ExcelWriter(out_path , engine='xlsxwriter')
df.to_excel(writer, sheet_name='Base', index=False )
writer.save()

#####
#PASO 3 -> LEVAN
#####
#Cargo un DataFram
base_bit = pd.read_excel("//bcbasv1156/canales_alternativo

#GUARDO COPIA PARA CONTAR ROWS LUEGO
base_bit.to_csv("L:/02-Tareas_programadas/Juanma/21-API/ro

#####
#PASO 4 BORRO SHEE
#####

#BORRO TODO SHEET
eraser = requests.delete(
    API_ENDPOINT + SHEET_NAME + "/" + "all",
    headers={"Authorization": f"Bearer {API_TOKEN}"}, "Cont
    verify=False
)

#####
#PASO 5 CARGO NUEV
#####

```

```

#####
#Defino una funció

def make_post_request(data):
    response = requests.post(
        API_ENDPOINT + SHEET_NAME,
        headers={"Authorization": f"Bearer {API_TOKEN}", "Content-Type": "application/json; charset=UTF-8"},
        data=json.dumps(data),
        verify=False
    )
    log_message = f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')} - [POST] {data['url']}"
    status_code = response.status_code
    description = status_descriptions.get(status_code, "Unknown")
    log_message += f"POST request: Status Code {status_code}, Description: {description}"

    if status_code == (200|203):
        logging.info(log_message)
    else:
        logging.error(log_message)

# Defino un diccionario con las descripciones de los códigos de respuesta
status_descriptions = {
    200: "OK",
    201: "Created",
    204: "No Content",
    400: "Bad Request",
    401: "Unauthorized",
    403: "Forbidden",
    404: "Not Found",
    500: "Internal Server Error",
}

# Configuro los ajustes de logging
log_file = "L:/02-Tareas_programadas/Juanma/21-API/status.log"
logging.basicConfig(filename=log_file, level=logging.INFO, encoding='utf-8')

# Defino el número de filas que quiero enviar en cada solicitud
ROWS_TO_SEND = 10

```

```
n_rows = 8000

# Itero sobre el DataFrame y envío cada n filas como un objeto JSON
for i in range(0, len(base_bit), n_rows):
    # Extraigo una parte del DataFrame
    df_part = base_bit.iloc[i:i+n_rows]
    # Convierto el DataFrame a JSON
    json_data = df_part.to_json(orient="records")
    # Convierto la cadena JSON a un objeto Python
    info_json = {"data": json.loads(json_data)}
    # Hago la solicitud POST y registro el estado de la respuesta
    make_post_request(info_json)
```

```
#####
#PASO 6 REQUEST Y
#####
```

```
sheets = requests.get(API_ENDPOINT + SHEET_NAME, headers={
    'Content-Type': 'application/json'})
data_sheets = sheets.json()
df_sheets = pd.DataFrame(data=data_sheets)
df_sheets.to_csv("L:/02-Tareas_programadas/Juanma/21-API/r")
```

```
#####
# SIGUE EN SCRIPT
#####
```

```
#### Cleaning the environment ####
```

```
# Remove all objects from the global environment.
rm(list = ls())

# Force R to garbage collect unused memory.
gc()
```

```

library(dplyr)
library(tidyverse)
library(tidyr)
library('DBI')
library(RDCOMClient)
library(htmlTable)
library(lubridate)

#### E-Mails ####

# Define the email addresses of the recipients of the alert
destinatario_to <- "-@bancociudad.com.ar; -@bancociudad.com"
#destinatario_cc <-
destinatario_cc <- " -@bancociudad.com.ar; -@bancociudad.com"
# Define the path where the text files containing the latest loaded dates of the bit
ImportPath <- "L:/02-Tareas_programadas/Juanma/21-API/rows_bit.txt"

#####
#          COMPARO ROWS
#####

# Read the text file containing the latest loaded dates of the bit
bit <- read.csv(paste0(ImportPath, "rows_bit.txt"), header = FALSE)

# Read the text file containing the latest loaded dates of the sheet
sheet <- read.csv(paste0(ImportPath, "rows_sheet.txt"), header = FALSE)

# Compare the number of rows in the two text files.
if (nrow(bit) == nrow(sheet)){
  # Get the last modification date of the file
  file_path <- "//bcbasv1156/canales_alternativos/Compartidos/Bitacora/Bitacora"
  file_modification_date <- format(file.info(file_path)$modified, "%d-%m-%Y %H:%M:%S")

  # Read the number of rows in the sheet file
  num_rows <- format(nrow(sheet) -1, big.mark = ".")
}

```

```

# Create the HTML body of the alert email.
html_cuerpo <- paste0("<html><p>
                        <font face='calibri'><font size = '3'
                        <br>
                        Estimados, buenos días. Se informa que

                        <i>Última modificación de archivo:</st
                        <i>Cantidad de filas en ps_bit.xlsx -

                        Saludos <br><br>
                        Powered by DMBC.
                        </font>
                        </p>
                        </html>")

# Create a COM object to interact with Outlook.
OutApp <- COMCreate("Outlook.Application")

# Create a new email message.
outMail = OutApp$CreateItem(0)

# Set the recipients of the email message.
outMail[["To"]] = destinatario_to
outMail[["CC"]] = destinatario_cc

# Set the subject of the email message.
outMail[["subject"]] = "*Alerta* Carga de bases bit [cor

# Set the HTML body of the email message.
outMail[["HTMLbody"]] = html_cuerpo

# Send the email message.
outMail$Send()

# Remove all objects from the global environment and for

```

```

#rm(list = ls())
#gc()

}else{

  # Create the HTML body of the alert email.
  html_cuerpo <- paste0("<html><p>
    <font face='calibri'><font size = '3'>
    <br>
    Estimados, buenos días.<br><br>
    Se informa que se ha registrado un err
    Saludos, <br><br><br>
    Powered by DMBC.<br><br><br>
    </font>
    </p>
  </html>")

  # Create a COM object to interact with Outlook.
  OutApp <- COMCreate("Outlook.Application")

  # Create a new email message.
  outMail = OutApp$CreateItem(0)

  #Attachment
  exportpath <- "//bcbasv1156/canales_alternativos/Compart

  outMail[["attachments"]]$Add(paste0(exportpath))

  # Set the recipients of the email message.
  outMail[["To"]] = destinatario_to
  outMail[["CC"]] = destinatario_cc

  # Set the subject of the email message.
  outMail[["subject"]] = "*ERROR* Carga de bases bit [corr

  # Set the HTML body of the email message.

```

```

outMail[["HTMLbody"]] = html_cuerpo

# Send the email message.
outMail$Send()

# Remove all objects from the global environment and for
#rm(list = ls())
#gc()
}

```

## ▼ DETALLE

En resumen, este script se utiliza para copiar datos de una hoja de cálculo de Google Sheets a un archivo de Excel, cargar nuevos datos desde un archivo de Excel a Google Sheets, y registrar el estado de las solicitudes durante el proceso. También se registra información sobre el número de filas en la hoja de cálculo antes y después de la carga de datos.

## Importación de bibliotecas

Se importan varias bibliotecas necesarias para el funcionamiento del script, incluyendo `requests` para hacer solicitudes HTTP, `pandas` para manejar datos en formato de tabla, `json` para trabajar con datos JSON, `logging` para registrar eventos, y `datetime` para manejar fechas y horas. También se desactivan las advertencias de seguridad de `urllib3`.

## Configuración de variables

Se configuran varias variables necesarias para el script, incluyendo el punto final (endpoint) de la API de Google Sheets, un token de autenticación, y el nombre de la hoja de cálculo a la que se accederá.

## PASO 1 - Realizar una solicitud a Google Sheets (Base original)

Se hace una solicitud GET a la API de Google Sheets para obtener datos de la hoja de cálculo especificada. Los datos se almacenan en un DataFrame de Pandas llamado `df`.

## PASO 2 - Generar una copia y guardarla como un archivo Excel

Se crea una copia del DataFrame `df` y se guarda en un archivo Excel en la ubicación especificada en `out_path`. Esto se hace utilizando la biblioteca `pandas` y `xlsxwriter`.

## PASO 3 - Cargar una hoja de cálculo de Excel

Se carga un archivo de hoja de cálculo de Excel desde una ubicación de red especificada y se almacena en un DataFrame llamado `base_bit`. También se guarda una copia de `base_bit` como un archivo de texto.

## PASO 4 - Borrar la hoja de cálculo actual

Se borra toda la hoja de cálculo actual en Google Sheets mediante una solicitud DELETE a la API. Esto elimina todos los datos de la hoja de cálculo.

## PASO 5 - Cargar un nuevo archivo a Sheets con registro de estados

Se define una función `make_post_request` que realiza una solicitud POST a la API de Google Sheets para cargar datos. Se registra el estado de la respuesta en un archivo de registro. El script itera sobre el DataFrame `base_bit`, envía los datos en lotes y registra el estado de cada solicitud.

## PASO 6 - Realizar una solicitud a Sheets y contar las filas

Se realiza otra solicitud GET a la API de Google Sheets para obtener datos de la hoja de cálculo después de cargar los nuevos datos. Los datos se almacenan en un nuevo DataFrame llamado `df_sheets`, y se guarda una copia de las filas en un archivo de texto.

## SIGUE EN SCRIPT R

El código menciona que continúa en un script R, lo que significa que este script de Python es parte de un flujo de trabajo más grande en el que se realiza algún procesamiento adicional en R.

▼ PLAYGORUND

```

#BASICS
#####
#           IMPORT
#####

import requests
import pandas as pd
import json

#####
# CARGO TOKEN Y E
#####

# Set the API endpoint URL
API_ENDPOINT = ""

# Set the API token
API_TOKEN = ""

# sheet ""

#####
# CONEXION A GSHE
#####

# Make a GET request to the API endpoint
response = requests.get(API_ENDPOINT + "psno7d1bp1me5", he
data = response.json()
df = pd.DataFrame(data=data)

# Create the data you want to upload
data = {
    "Prueba Pablo": "Lucho Tigani",
    "Prueba Pablo 2": 35
}

```

```

#GENERO VARIABLES
columna = "Prueba Pablo"
valor = "Lucho Tigani"

# Make a POST request to the API endpoint
#CARGAR INFO
response = requests.post(
    API_ENDPOINT + "psno7d1bp1me5",
    headers={"Authorization": f"Bearer {API_TOKEN}", "Content-Type": "application/json; charset=UTF-8"},
    data=json.dumps(data),
    verify=False
)

#BORRA TODO LO QUE CUMPLE LA CONDICION
response_2 = requests.delete(
    API_ENDPOINT + "psno7d1bp1me5"+ "/" + columna + "/" +
    headers={"Authorization": f"Bearer {API_TOKEN}", "Content-Type": "application/json; charset=UTF-8"},
    verify=False
)

#BORRA TODO
response_3 = requests.delete(
    API_ENDPOINT + "psno7d1bp1me5"+ "/" + "all",
    headers={"Authorization": f"Bearer {API_TOKEN}", "Content-Type": "application/json; charset=UTF-8"},
    verify=False
)

#####
#          EXCEL
#####

#CARGO UN DF
nuevo_dataframe = pd.read_excel("L:/02-Tareas_programadas/

```

```
# Convert the DataFrame to JSON
json_data = nuevo_dataframe.to_json(orient="records")

info_json = {
    "data": json.loads(json_data) # Convert the JSON string to a Python dict
}

response_4 = requests.post(
    API_ENDPOINT + "psno7d1bp1me5",
    headers={"Authorization": f"Bearer {API_TOKEN}", "Content-Type": "application/json"},
    data=json.dumps(info_json),
    verify=False
)
```

## ▼ Comentarios:

- Se debe completar el nombre de las columnas, caso contrario:  
{"error":"Spreadsheet is empty. Fill first row with the column names, each next row will be treated as a record."}
- Se pueden subir hasta **8000 por cada post request**
- Límite de **celdas** en sheets 10.000.000

## ▼ Lo que se viene ...

AMPLIACIÓN A OTRAS BASES PRODUCTIVAS