A brief technical overview of a process in development for TFNs equipment monitoring:

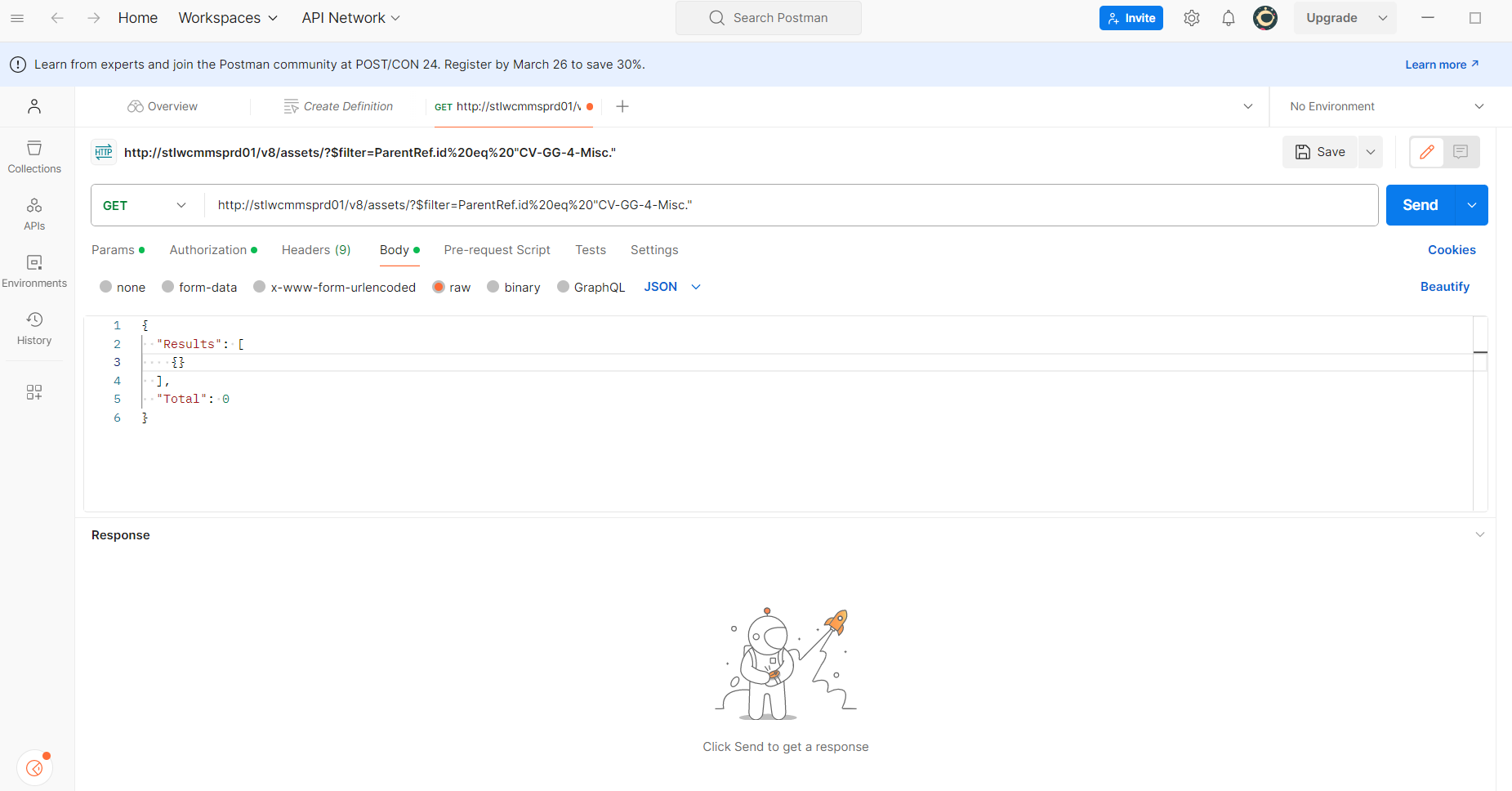
Back-end interaction with Maintenance Connection, and automated reporting.

LINK to API DOCUMENTATION: [API Docs (maintenanceconnection.com)](https://api.maintenanceconnection.com/v8/help/docs)

The Maintenance Connection Web API is a RESTful service with each resource accessible through a unique URI using HTTP verbs.

Meaning you’ll need to make an HTTP request and use json structures to further specify the request.

There are a few ways to do this. I use Postman and Python. Postman is good to verify requests, and experiment with API interaction.   
I use Python to build applications around requests validated in Postman (shown below).



Let’s look at the parameters for making valid requests.  
I use the json, and requests libraries. These libraries take url and headers, send the request, and capture the results from the request. Example below:

import os

import time

**import json**

**import requests**

import pandas as pd

**headers = {**

**'Accept': 'application/json',**

**'Content-Type': 'application/json',**

**'Authorization': 'Basic KEY GOES HERE',**

**'Connection': 'keep-alive'**

**}**

desktop\_dir = os.path.expanduser(r"C:\Users\USER\OneDrive - Bayer\Desktop")

# List of lab IDs

labs = ['4101', '4103', '4105', '4121', '4123', '4125', '4130', '4141', '4143', '4145', '4153', '4251', '4301', '4303', '4305', '4320-1', '4321', '4323', '4329', '4340', '4341', '4343', '4345', '4401', '4404', '4406', '4408', '4424', '4426', '4428', '4444', '4446', '4448', '3220']

# Collecting data

all\_data = []

load = ''

for lab in labs:

url = f'http://**stlwcmmsprd01/v8**/assets?$filter=ParentRef.ID%20eq%20"CV-GG-4-{lab}"' if lab != '3220' else f'http://stlwcmmsprd01/v8/assets?$filter=ParentRef.ID%20eq%20"CV-GG-3-3220"'

try:

**response = requests.get(url, headers=headers) # GET request**

if response.status\_code == 200:

load += '\*'

print('\r'+ load, end='')

data = response.json()

results = data['Results']

for item in results:

**all\_data.append({**

**'LabID': lab, # Include LabID in the data**

**'PK': item.get('PK', None),**

**'ID': item.get('ID', None),**

**'Name': item.get('Name', None),**

**'Model': item.get('Model', None),**

**'UDFChar4': item.get('UDFChar4', None),**

**'UDFDate4': item.get('UDFDate4', None),**

**'UDFDate3': item.get('UDFDate3', None)**

**})**

else:

print(f"Error: Request for lab {lab} failed with status code {response.status\_code}")

except Exception as e:

print(f"An error occurred while processing lab {lab}: {str(e)}")

# Convert to DataFrame

df = pd.DataFrame(all\_data)

# Define file path

file\_path = os.path.join(desktop\_dir, "lab\_data.xlsx")

# Save to Excel with each lab on a separate sheet

with pd.ExcelWriter(file\_path, engine='openpyxl') as writer:

unique\_labs = df['LabID'].unique()

for lab in unique\_labs:

lab\_df = df[df['LabID'] == lab]

lab\_df.to\_excel(writer, sheet\_name=str(lab), index=False)

print(f"\nData has been saved to '{file\_path}', with separate sheets for each lab.")

For the script to work. The correct headers need to be defined in json format.

Then provide my API key.

The script loops through each lab ID and constructs a URL to fetch data from the web service.

It then makes HTTP GET requests to these URLs using the requests library. Upon receiving a response, it checks if the request was successful (status code 200) and extracts relevant data from the response JSON.

The base url is: stlwcmmsprd01/v8/

Note the following filers: assets?$filter=ParentRef.ID%20eq%20"CV-GG-4-{lab}

In this example {lab} is a placeholder so to iterate through the list of labs and make multiple requests in one run.

Assets can be pinged by using relevant tags.

In this case it’s the ParentRef.ID or Parent:(as shown in MC)

Assets can also be easily tracked with the unique PK values.

If we look at the json body. We see the PK is requested from an asset.

We request by formatting our url and response body. Which can be tested first in Postman.

The script pulls the data and structures into a DataFrame that will become an excel report.

What I’m getting from each asset is PK, ID, Name, Serial#, Calibration date, and next Calibration date.

Note that this is just an example and not from my main project. The output from this example is below:

