# Cracking Nafta’s Reactor Simulation

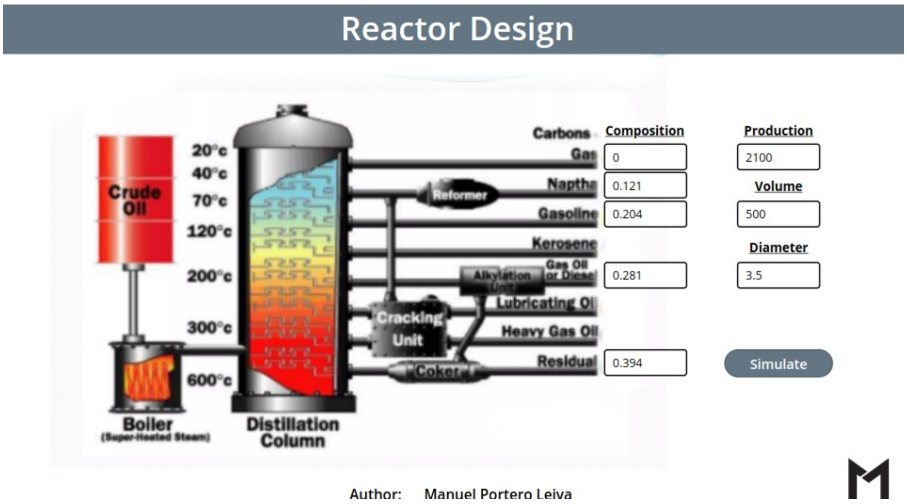
PowerApp Documentation

31/12/2022

**Author:** Manuel Portero Leiva

# Introduction

This document has the purpose to explain the different parts of the Reactor Designing Function App, its code and functionalities, for understanding and replication purposes. The different parts of the architecture solution are show below.



Reactor Design

Composition

Gas: 0

Naptha: 0.121

Gasoline: 0.204

Gas Oil or Diesel: 0.281

Residual: 0.394

Production: 2100

Volume: 500

Diameter: 3.5

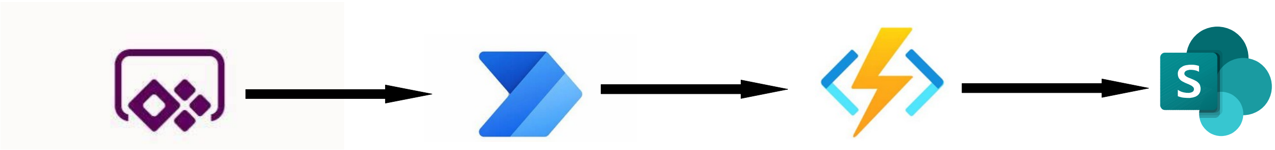
Author: Manuel Portero Leiva

**Picture 1:** Cracking Nafta Reactor Simulation Layout

# Architecture

The composition of the architecture starts in the PowerApp. Once the Reactor design is choosen and the calculate button is pressed, a Function app is triggered via powerAutomate and a Sharepoint list is filled with the reactor’s design data.

A full diagram of the solution is shown below.

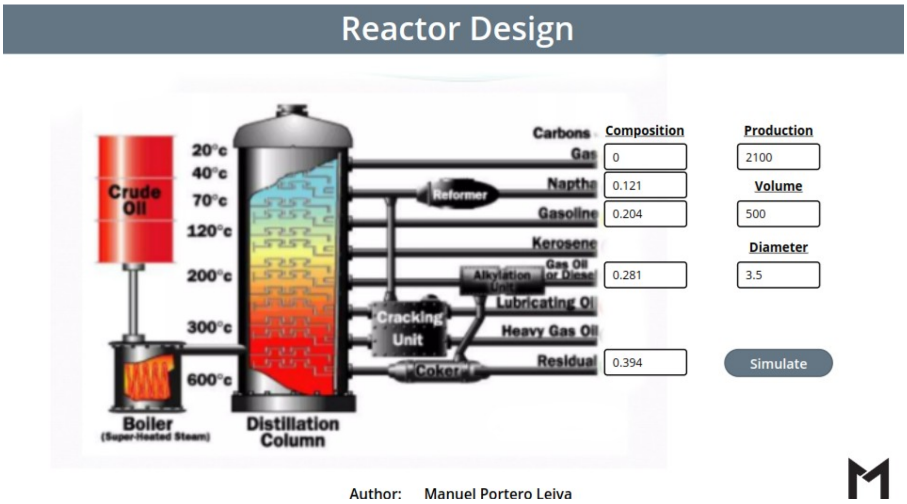


**Picture 2:** Cracking Nafta Reactor Simulation Architecture

# PowerApp

## Main Screen

The main screen is composed by the main landscape picture and the navigation buttons to the others screens.



Reactor Design

| Carbons | Composition |
|---|---|
| Gas | 0 |
| Naptha | 0.121 |
| Gasoline | 0.204 |
| Gas Oil or Diesel | 0.281 |
| Residual | 0.394 |

Production: 2100

Volume: 500

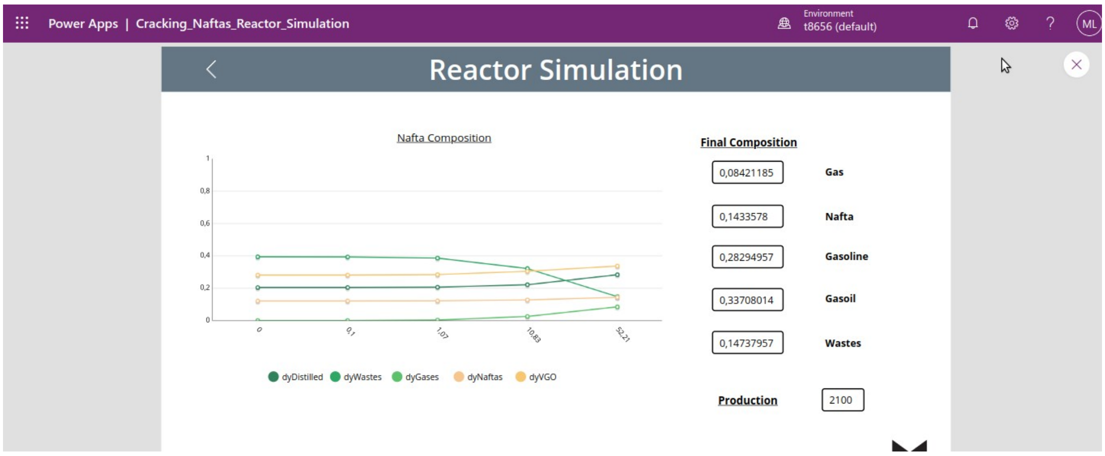
Diameter: 3.5

Author: Manuel Portero Leiva

**Picture 4:** Cracking Nafta Reactor Simulation  main Screen

## Details Screen

The Details Screen is composed by the different Reactor parameter’s and a reactor design diagram.



Final Composition

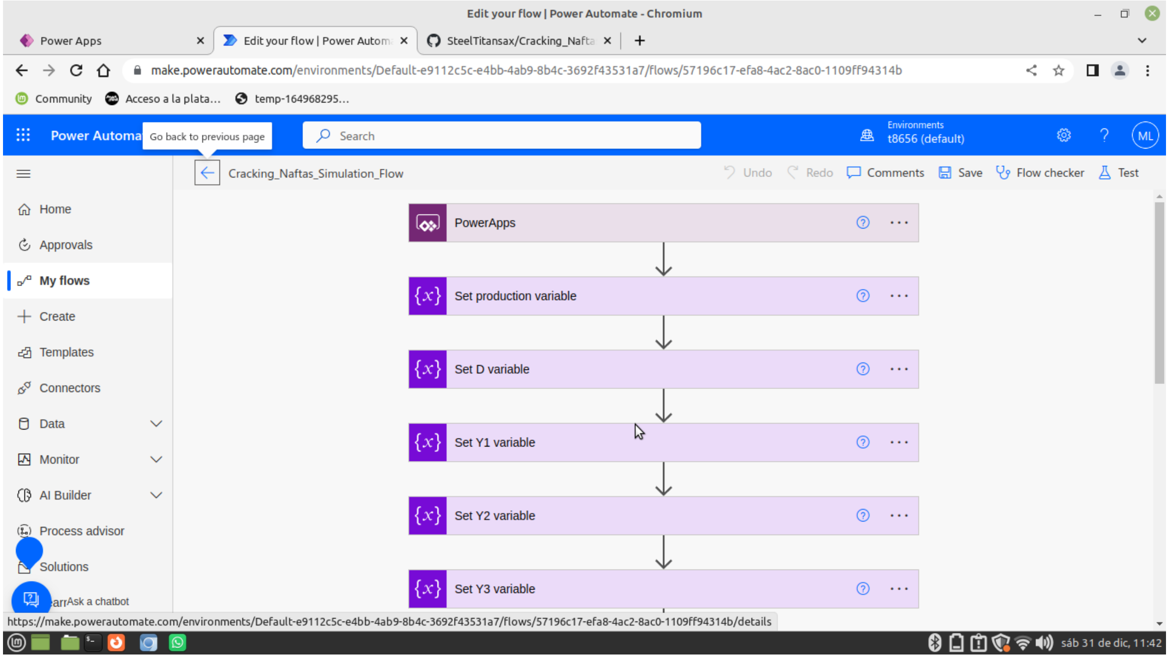
| | |
|---|---|
| 0,08421185 | Gas |
| 0,1433578 | Nafta |
| 0,28294957 | Gasoline |
| 0,33708014 | Gasoil |
| 0,14737957 | Wastes |

Production: 2100

**Picture 5:** Details Screen

# PowerAutomate Flow

The PowerAutomate flow receive the paremeters from the PowerApp clear, the Sharepoint list, call the Azure function with an Http Request action and finally fill the Sharepoint list with the new design parameters.


Edit your flow | Power Automate - Chromium
Power Apps
Edit your flow | Power Autom
SteelTitansax/Cracking_Nafta
make.powerautomate.com/environments/Default-e9112c5c-e4bb-4ab9-8b4c-3692f43531a7/flows/57196c17-efa8-4ac2-8ac0-1109ff94314b
Community
Acceso a la plata...
temp-164968295...
Power Automa
Go back to previous page
Search
Environments
t8656 (default)
ML
Cracking_Naftas_Simulation_Flow
Undo
Redo
Comments
Save
Flow checker
Test
Home
Approvals
My flows
Create
Templates
Connectors
Data
Monitor
AI Builder
Process advisor
Solutions
Ask a chatbot
PowerApps
Set production variable
Set D variable
Set Y1 variable
Set Y2 variable
Set Y3 variable
https://make.powerautomate.com/environments/Default-e9112c5c-e4bb-4ab9-8b4c-3692f43531a7/flows/57196c17-efa8-4ac2-8ac0-1109ff94314b/details
sáb 31 de dic, 11:42

**Picture 6:** Setting Screen

# Azure Function

The Reactor Design azure function will receive the parameters from the PowerAutomate flow and will calculate the Volume and other design parameters of the choosen reactor. The ecuations design code for each reactor type are shown below

### Cracking Nafta’s Reactor:

```
#Libraries
import logging
import azure.functions as func
import json
import numpy as np
from scipy.integrate import solve_ivp


def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('Reactor designing Begins...')
```

```
try:
    req_body = req.get_json()
except ValueError:
    pass
else:
 production = req_body.get('production') # petrol barrels per day
    production = float(production)
    V0 = production * 0.006624470622 # m3/h
    density = 0.715 #g/cm3

    massic_flow_0 = V0  * density * 100**3 #g/h
    D = req_body.get('D') #m

    #      res , vgo , des , naf , gas
    y1 = req_body.get('y1') # initial non-processed petrol composition
    y2 = req_body.get('y2') # initial non-processed petrol composition
    y3 = req_body.get('y3') # initial non-processed petrol composition
    y4 = req_body.get('y4') # initial non-processed petrol composition
    y5 = req_body.get('y5') # initial non-processed petrol composition

    # Parsing input data

    D = float(D)
    y1 = float(y1)
    y2 = float(y2)
    y3 = float(y3)
    y4 = float(y4)
    y5 = float(y5)


    y0 = [y1,y2,y3,y4,y5] # initial non-processed petrol composition


    k1 = 0.147
    k2 = 0.022
    k3 = 0.020
    k4 = 0.098
    k5 = 0.057
    k6 = 0.007
    k7 = 0
    k8 = 0.003
    k9 = 0
    k10 = 0

    V_init = 0
    V_final = 500

    def reactionSystem(V,y):
        y1 = y[0]
        y2 = y[1]
        y3 = y[2]
```

```
            y4 = y[3]

            dyWastes = (-k1*y1-k2*y1-k3*y1-k4*y1)/V0
            dyVGO = (k1*y1-k5*y2-k6*y2-k7*y2)/V0
            dyDistilled = (k2*y1+k5*y2-k8*y3-k9*y3)/V0
            dyNaftas = (k3*y1+k6*y2-k8*y3-k10*y4)/V0
            dyGases = (k4*y1+k7*y2+k9*y3+k10*y4)/V0

            return np.array([dyWastes, dyVGO,dyDistilled,dyNaftas,dyGases])


        sol = solve_ivp(reactionSystem, (V_init,V_final),y0)

        # Consolidating outputs:
        dyWastes = sol.y[0]
        dyVGO = sol.y[1]
        dyDistilled = sol.y[2]
        dyNaftas = sol.y[3]
        dyGases = sol.y[4]
        t = sol.t

        sol_json = []

        for item in range(5):

            sol_details = {
                    "dyWastes": dyWastes[item],
                    "dyVGO": dyVGO[item],
                    "dyDistilled": dyDistilled[item],
                    "dyNaftas": dyNaftas[item],
                    "dyGases": dyGases[item],
                    "t": t[item]
                }

            sol_json.append(sol_details)

        logging.info(sol_json)

        return json.dumps(sol_json)

    return func.HttpResponse("Reactor design succesfully...",status_code=200)
```