Reactor Designing Function App

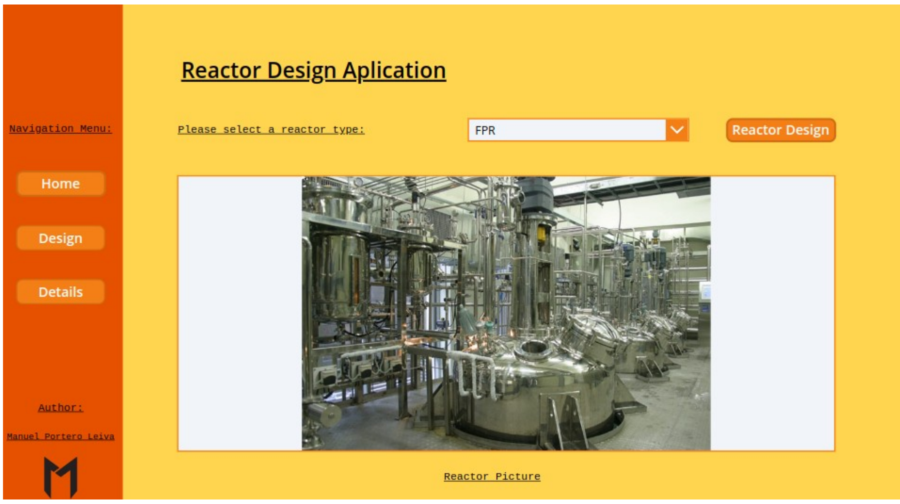
PowerApp Documentation

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.

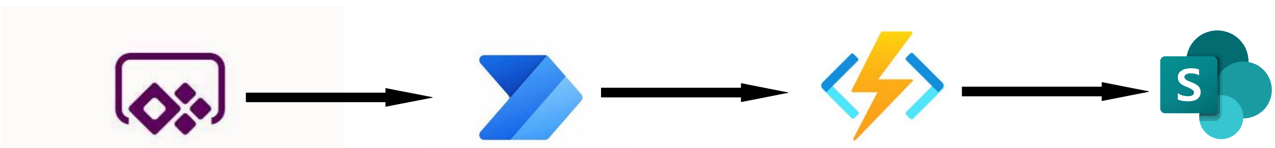


Picture 1: Reactor Designing Function App Layout

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.

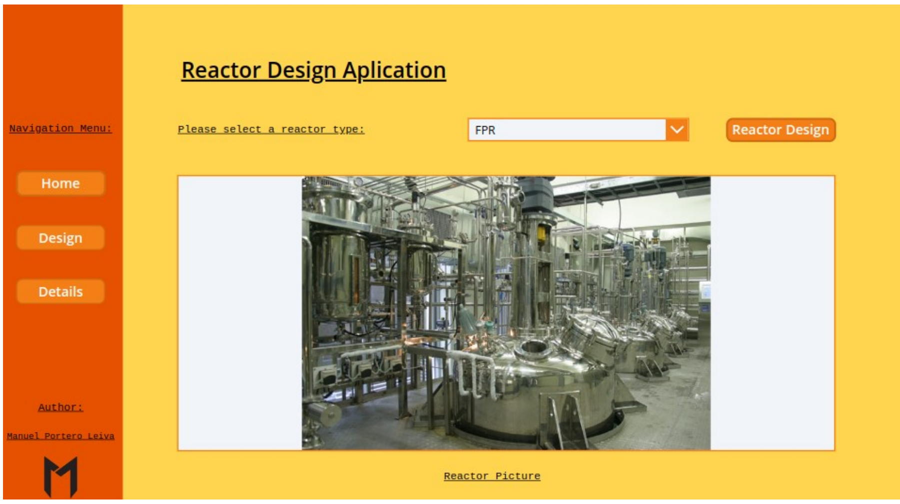


Picture 2: Reactor Designing Function App solution Architecture

PowerApp

Main Screen

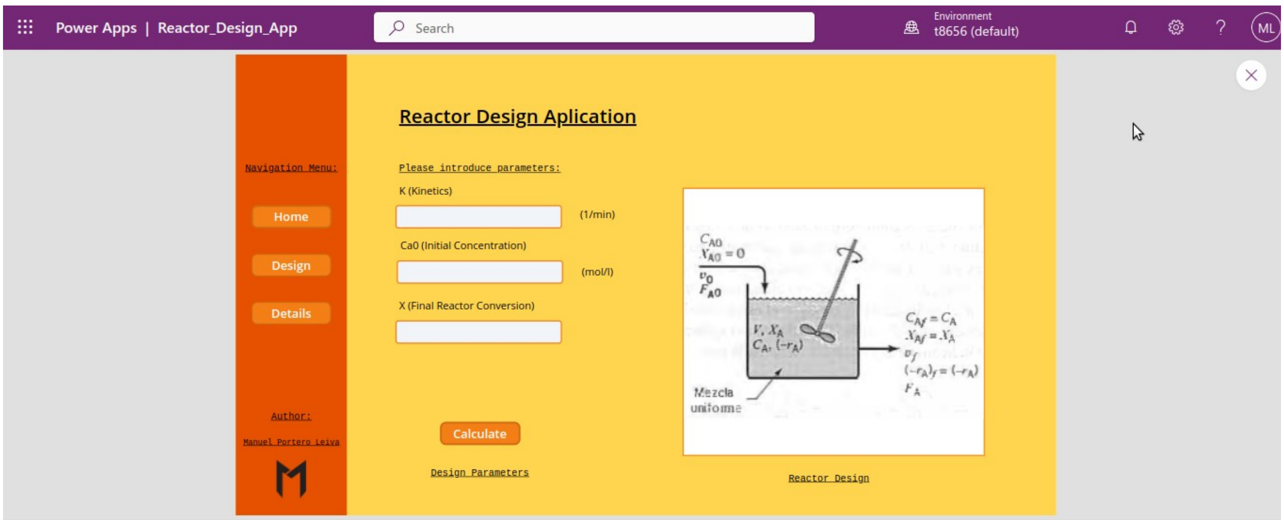
The main screen is composed by the main landscape picture and the navigation buttons to the others screens.



Picture 4: Reactor Designing Function App main Screen

Reactor Design Screen

The Reactor Design Screen is composed by the different Reactor parameter's text inputs and a calculate button. Once the calculate button is pressed the reactor design flow is activated.



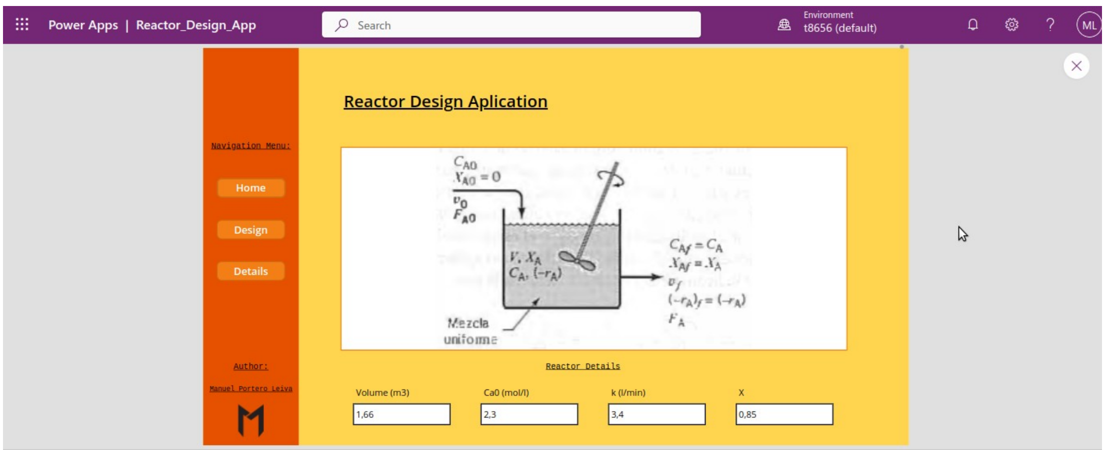
Picture 5: Reactor Design Screen

The code of the calculate button is the following one :

```
Reactor_Designing_App_Flow.Run(
    KTextInput.Text;
    V0TextInput.Text;
    Ca0TextInput.Text;
    XTextInput.Text;
    1
).result = true;;
Refresh(CSTR_Reactor);;
Refresh(CSTR_Reactor);;
Navigate(Details);;
Notify(
    "Reactor Designed Successfully";
    Success
);;
Reset(V0TextInput);;
Reset(Ca0TextInput);;
Reset(XTextInput);;
Reset(KTextInput)
```

Details Screen

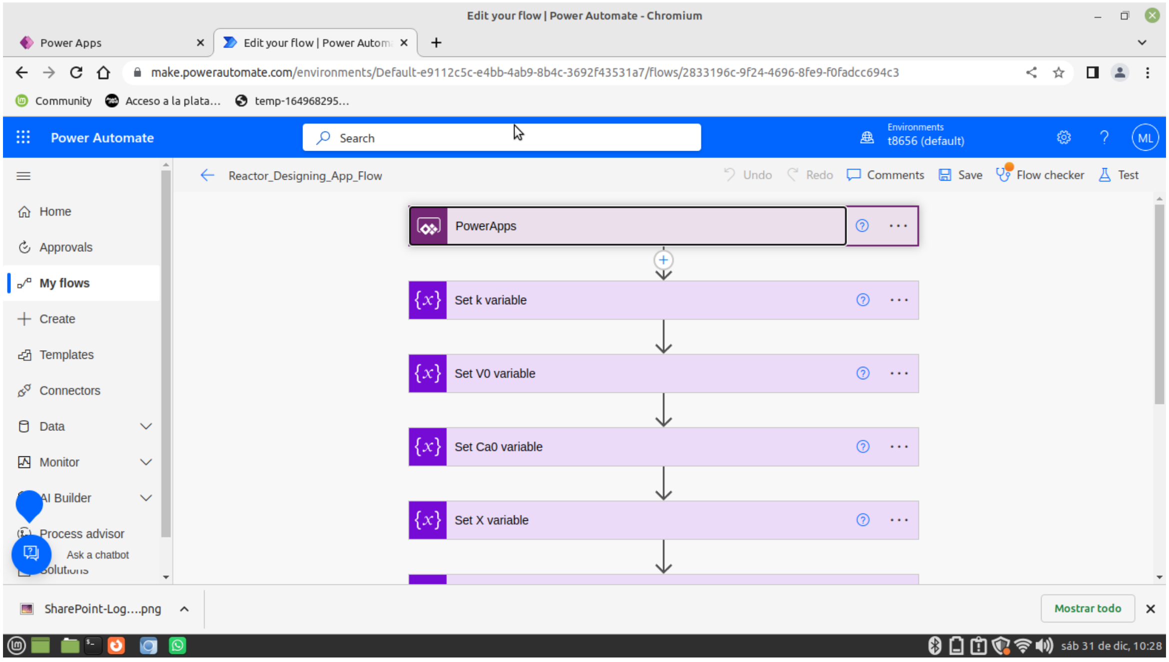
The Details Screen is composed by the different Reactor parameter’s and a reactor design diagram.



Picture 6: 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.



Picture 6: Setting Screen

<u>(Note:This flow has a switch calling different reactor types, the end to end demo is only ready for a CSTR reactor)</u>

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

CSTR Reactor:

```
import logging
import azure.functions as func
import json

def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('Reactor designing Begins...')
    try:
        req_body = req.get_json()
    except ValueError:
        pass
    else:
```

```
    #Variable declarations
        #---------------------
        k = req_body.get('k')#1/min
        logging.info('k: ' + k)
        Ca0 = req_body.get('Ca0')#mol/l
        logging.info('Ca0: ' + Ca0)
        X = req_body.get('X')
        logging.info('X: ' + X )


       #Parsing
        logging.info("Parsing variables ... ")

        k = float(k)
        Ca0 = float(Ca0)
        X = float(X)

        logging.info("Variables parsed")

        #Kinetics
        #--------
        def rA(X):
            Ca = Ca0*(1-X)
            return k*Ca

        #Design equation of an CSTR
        #--------------------------
        V = (-Ca0 * X) /-rA(X)
        logging.info("Volume : " + str(V) )

return json.dumps(
        {
            "Volume": V,
            "Ca0": Ca0,
            "rA": rA(X),
            "k" : k,
            "X" : X
        }
    )
 return func.HttpResponse("Reactor design succesfully...",status_code=200)
```

DSTR Reactor:

```
#Libraries
import logging
from scipy.integrate import quad
import azure.functions as func
```

```
import json


def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('Reactor designing Begins…')
    try:
        req_body = req.get_json()
    except ValueError:
        pass
    else:
    # Variable declarations
        # ---------------------
        k = 0.3  # 1/min
        logging.info('k: ' + k )
        Ca0 = 10  # mol/l
        logging.info('Ca0: ' + Ca0 )
        X = 0.8
        logging.info('X: ' + X )

        # Kinetics
        # --------
        def rA(X):
            Ca = Ca0 * (1 - X)
            return k * Ca

        def integral(X):
            return X / rA(X)



        IntergerResult, err = quad(integral, 0, X)
        T = Ca0 * IntergerResult # hours
        logging.info("Reaction time : " + str(T))
```

```
        return json.dumps(
        {
            "Volume": T,
            "Ca0": Ca0,
            "rA": rA(X),
            "k" : k,
            "X" : X
        }
        )
    return func.HttpResponse("Reactor designed successfully")
```

DSTR Reactor:

```
#Libraries
import logging
from scipy.integrate import quad
import azure.functions as func
import json

def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('Reactor designing Begins...')
    try:
        req_body = req.get_json()
    except ValueError:
        pass
    else:
        #Variable declarations
        #---------------------
        k = req.body('k') #1/min
        logging.info('k: ' + k )
        Ca0= req.body('Ca0') #mol/l
```

```
        logging.info('Ca0: ' + Ca0 )
        V0 = req.body('V0') #l/min
        logging.info('V0: ' + V0 )
        X = req.body('X')
        logging.info('X: ' + X )
        Fa0 = Ca0 * V0 #mol/min
        logging.info('Fa0: ' + Fa0 )

        #Kinetics
        #--------
        def rA(X):
            Ca = Ca0*(1-X)
            return -k*Ca
        #Design equation of an FPR
        #-------------------------
        def integral(X):
            return Fa0/-rA(X)

        V,err = quad(integral,0,X)
        logging.info("Volumen : " + str(V) )

        return json.dumps(
        {
            "Volume": V,
            "Ca0": Ca0,
            "rA": rA(X),
            "k" : k,
            "X" : X
        }
        )
    return func.HttpResponse("Reactor designed successfully")
```