RPA Monitoring Tool

PowerApp Documentation

18/10/2023

Author: Manuel Portero Leiva

Introduction

This document has the purpose to explain the different parts of the RPA Monitoring Tool, its code and functionalities, for understanding and replication purposes. The different parts of the architecture solution are show below.

Picture 1: RPA Monitoring Tool Main Layout

Architecture

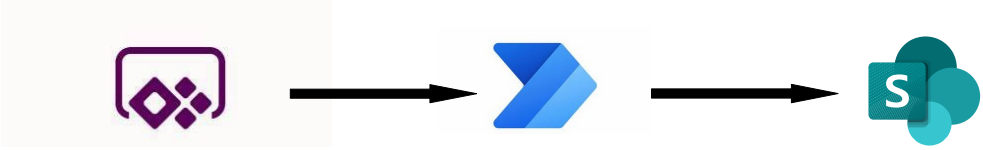
The composition of the architecture starts in the PowerApp. The different parts of the app triggers different Http request the UiPath Ochestrator API via PowerAutomate and the requested information is stored in different Sharepoint lists.

The RPA app functionalities are :

- **KPI Screen :** A screen showing the different KPI of our UiPath RPA environment.
- **Scheduled Jobs Screen :** A screen showing the processes that are goin to run today in the next ours
- **Error LogsScreen :** A screen showing all today's the errors categorized by robots.
- **TroubleShooting Screen :** A screen displaying the different errors troubleshooting categorized by Robot Message and Error Type.
- **Communication Screen :** A screen used to comunicate via email with the RPA process owners.

- **Action Screen :** A screen set up for perform actions in the RPA environment (Restart machines, set new azure devops tasks etc,...)

A full diagram of the solution is shown below.



Picture 2: RPA Monitoring Tool Architecture

(Improve CRUD codes)

PowerApp

KPI Screen

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

Picture 3: KPI Screen Layout RPA Monitoring Tool

The KPI of this screens are the number of today's jobs, number of machines in the system, number of failed processes, number of robots of the system and the number of licenses. A part of that this screen has a heat map where we can truck easily which process of the system is failing.

The code for calculating each KPI and for the heat map is shown below :

Number Jobs Today : CountRows(Distinct(RPA_ScheduledProcess;Title))

Number of Machines : CountRows(RPA_Machines)

Num Process Failed: CountRows(Distinct(RPA_DailyErrorLogs;ProcessName))

Num Robots : CountRows(RPA_Robots)

Num Licenses : CountRows(RPA_Robots)

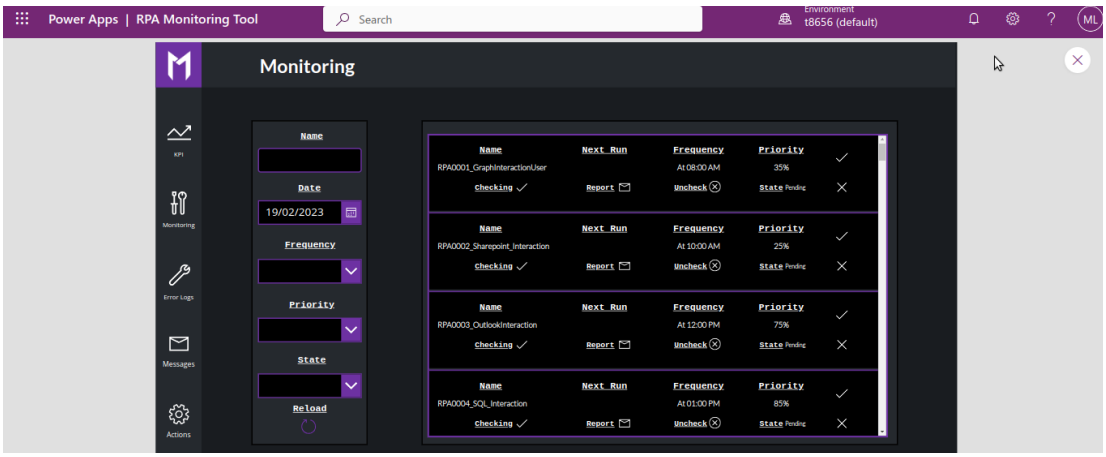
Heat Map (Fill Property) : If(LookUp(RPA_DailyErrorLogs;"RPAxxxx

"in ProcessName;true);Red;Green)

Monitoring Screen

The Monitoring Screen is composed by the MonitoringGallery (connected to the RPA_ScheduledProcesses Sharepoint List) , a textInput and dropdowns for filtering by processName, date , frequency, priority , state and a reload button. In side the gallery we have the check (‘yes’) button, check (‘no’) button , the (‘report’) button and the (‘checking’) button.

The report button will send a generic email to the process owner alerting something wrong is happening with his robot.



Picture 4: Monitoring Screen

The code of the different screen components is shown below :

- MonitoringGallery (default property) :

(note : it remains to add the date filter but i didn’t code it for demo purposes).

```
Filter(
    Search(
        RPA_ScheduledProcess;
        TextNameInput.Text;
        "Title"
    );
    If(
        FrequencyDropdown.SelectedText.Value = Blank();
        true;
        StartProcessCronSummary = FrequencyDropdown.SelectedText.Value
    ) && If(
        PriorityDropdown.SelectedText.Value = Blank();
        true;
        JobPriority = PriorityDropdown.SelectedText.Value
    ) && If(
        StateDropdown.SelectedText.Value = Blank();
        true;
        State = StateDropdown.SelectedText.Value
    )
)
```

CheckIcon(OnSelect property):

```
Patch(
    RPA_CheckedProcesses;
    Defaults(RPA_CheckedProcesses);
    {
        Title: ThisItem.Title;
```

```
        JobPriority: ThisItem.JobPriority;
        StartProcessCronSummary: ThisItem.StartProcessCronSummary;
        StartProcessNextOccurrence: ThisItem.StartProcessNextOccurrence;
        Status: "Success"
    }
);;Notify("ProcessName Checked Successfully";Success)
```

CheckFailedIcon(OnSelect property):

```
Patch(
    RPA_CheckedProcesses;
    Defaults(RPA_CheckedProcesses);
    {
        Title: ThisItem.Title;
        JobPriority: ThisItem.JobPriority;
        StartProcessCronSummary: ThisItem.StartProcessCronSummary;
        StartProcessNextOccurrence: ThisItem.StartProcessNextOccurrence;
        Status: "Failed"
    }
);;Notify("ProcessName Checked as failed Successfully";Success)
```

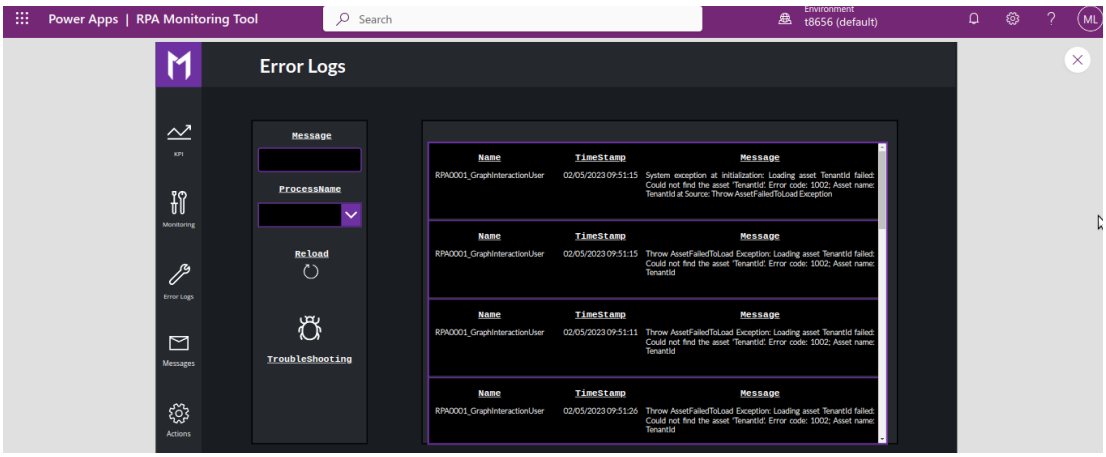
StatusLabel (Text Property):

```
If(
    LookUp(
        RPA_CheckedProcesses;
                (Nombre = ThisItem.Nombre && StartProcessNextOccurrence =
ThisItem.StartProcessNextOccurrence && Status = "Success");
        true
    );
    "Checked";
    If(
```

```
        LookUp(
            RPA_CheckedProcesses;
                (Nombre = ThisItem.Nombre && StartProcessNextOccurrence = ThisItem.StartProcessNextOccurrence && Status = "Failed");
            true
        );
        "Failed";
        "Pending"
    )
)
```

Error Logs Screen :

The ErrorLogs Screen is composed by the ErrorLogsGallery (connected to the RPA_ScheduledProcesses Sharepoint List) , a textInput, a dropdown of processName , a reload button and a troubleshooting navigation button.



Picture 5: ErrorLogs Screen

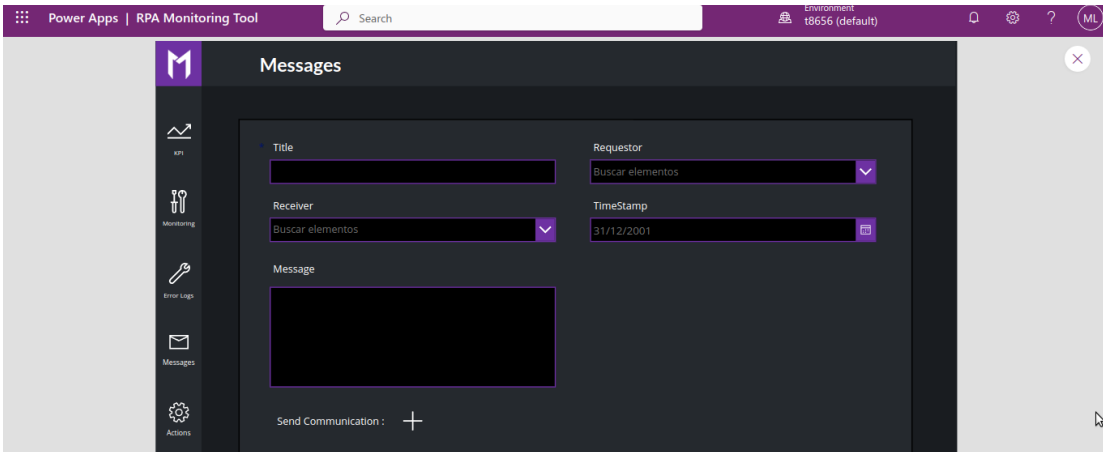
The code of the different screen components is shown below :

- ErrorLogsGallery (default property) :

```
Filter(
    Search(
        RPA_DailyErrorLogs;
        MessageTextInput.Text;
        "Message"
    );
                    If(ProcessNameDropdown.Selected.ProcessName = Blank();true;ProcessName = ProcessNameDropdown.Selected.ProcessName)
)
```

Comunication screen :

The Communication Screen is composed by a communication form and a new communication button. Once the new communication button is pressed, an email is sent to the selected person with the form information :



Picture 6: TroubleShooting Screen

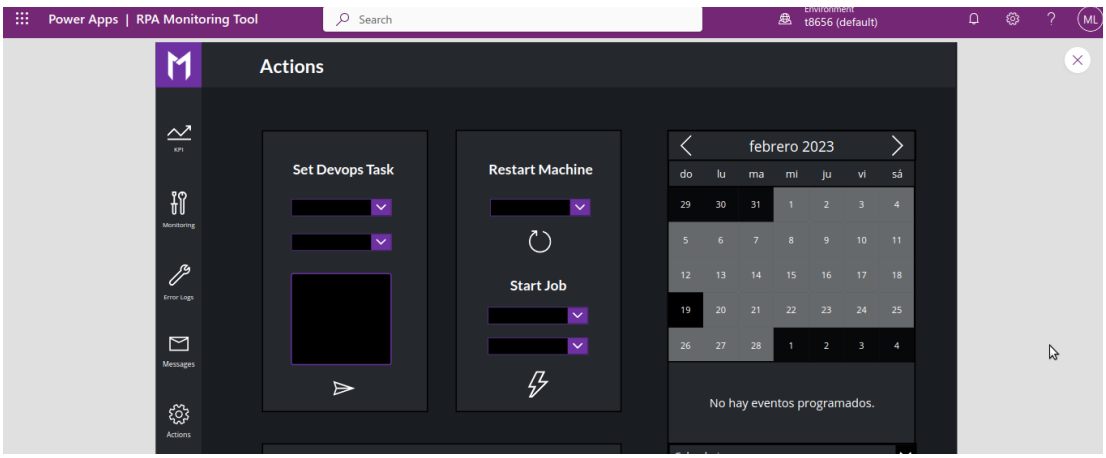
The code of the different screen components is shown below :

New Communication button (onSelect property):

```
If(
    SubmitForm(CommunicationForm);;
    NewForm(CommunicationForm);
    RPA_Monitoring_Tool_Communication.Run(
        DataCardValue7.Text;
        DataCardValue8.Selected.Email;
        DataCardValue9.Selected.Email;
        Now();
        DataCardValue11.Text
    );
    Notify(
        "Message sent successfully";
        Success
    );
    Notify(
        "Message error failed";
        Error
    )
)
```

Actions Screen

The Communication Screen is composed by Set devops activity section a restart machine section ,a start job section and a calendar (for each parts we set up the diferent dropdowns and textinputs , and after press the buttons of each parts, powerAutomate emails are triggered).



Picture 7: Actions Screen

The code of the different screen components is shown below :

Set devops task button (onSelect property):

```
If(
    RPA_Monitoring_Tool_Devops.Run(
        AssignedToDropdown.SelectedText.Value;
        DevopsKindDropdown.Selected.displayName;
        DevopsText.Text
    ).result = "true";
    Reset(AssignedToDropdown);;
    Reset(DevopsKindDropdown);;
    Reset(DevopsText);;
    Notify(
        "Task Added Successfully";
        Success
    );
    Reset(AssignedToDropdown);;
    Reset(DevopsKindDropdown);;
    Reset(DevopsText);;
    Notify(
        "Error Adding Task";
        Error
    )
)
```

Set devops task button (onSelect property):

```
If(
    RPA_Monitoring_Tool_Machines.Run(Machine2Dropdown.SelectedText.Value).r
esult="true";
    Reset(Machine2Dropdown);;
    Notify(
```

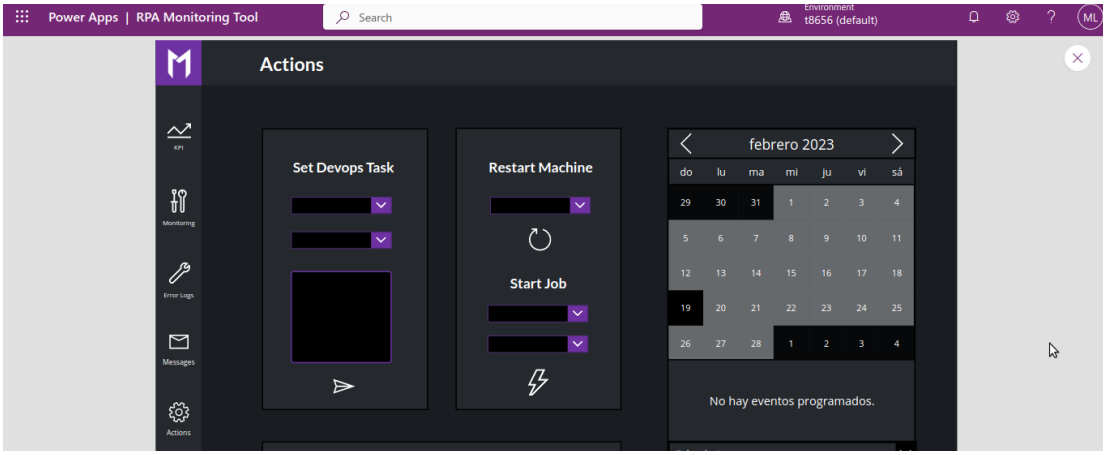
```
        "Machine Restarted Successfully";
        Success
    );
    Reset(Machine2Dropdown);;
    Notify(
        "Error Restarting Machine";
        Error
    )
)
```

PostJob Button :

```
Reset(JobDropdown);; RPA_Monitoring_Tool_PostJob.Run(
        JobDropdown.SelectedText.Value;
        MachineDropdown.Selected.displayName)
    ;;Notify("Machine Restarted Successfully";Success)
```

Troubleshooting Screen:

The TroubleShooting Screen is composed by a TroubleShootingGallery and process type and error type dropdowns, a part of that we can find a reload and a create new troubleshooting error buttons. Inside the gallery we can find CRUD operation buttons (Edit Delete and Details).



Picture 8: Monitoring Screen

The code of the different screen components is shown below :

TroubleShootingGallery (OnSelect property) :

```
Filter(
    Search(
        RPA_Troubleshooting;
        MessageTextInput_1.Text;
        "Message"
    );
    If(
        ProcessNameDropdown_1.Selected.Título = Blank();
        true;
        "ProcessName" = ProcessNameDropdown_1.Selected.Título
    );
    If(
        ErrorTypeDropdown.Selected.'Error Type' = Blank();
        true;
        "Error Type" = ProcessNameDropdown_1.Selected.'Error Type'
    )
)
```

Create new TroubleShooting Error (onSelect property) :

```
Navigate(CreateTroubleShooting);;NewForm(CreateErrorForm)
```

Edit TroubleShooting Error (onSelect property):

```
Select(Parent);;EditForm(EditErrorForm);;Navigate(EditTroubleShooting)
```

View Details TroubleShooting Error (onSelect property):

```
Select(Parent);;ViewForm(ViewErrorForm);;Navigate(ViewTroubleShooting)
```

Delete TroubleShooting Error (onSelect property) :

```
Remove(RPA_Troubleshooting; ErrorLogsGallery_1.Selected);; If (IsEmpty(Errors(RPA_Troubleshooting; ErrorLogsGallery_1.Selected));"");;Notify("Error deleted successfully";Success)
```

Reload Button (onSelect property):

Reset(MessageTextInput_1);;Refresh(RPA_DailyErrorLogs);;Reset(ErrorLogsGallery_1);;Notify("All filters reset";Success)

CreateNewTroubleShooting Error → **Create Error Button (onSelect property):**

SubmitForm(CreateErrorForm);;NewForm(CreateErrorForm);;Notify("Error reported created successfully";Success);;Navigate(TroubleShooting)

EditTroubleShooting Error → **Edit Error Button (onSelect property):**

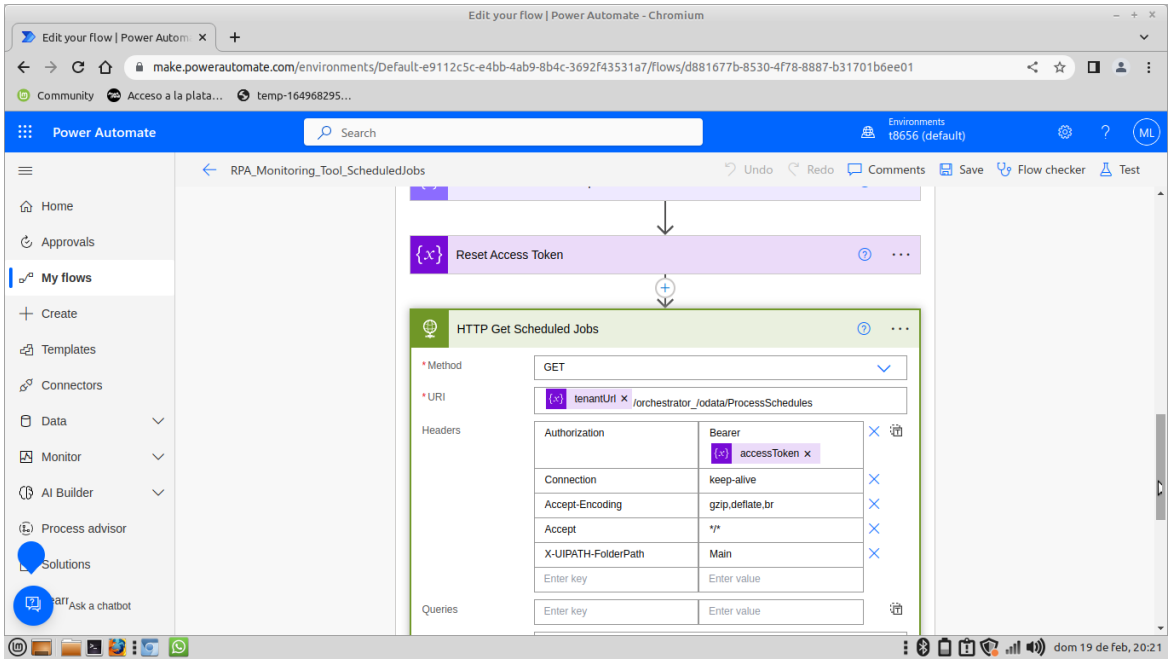
SubmitForm(CreateErrorForm);;NewForm(CreateErrorForm);;Notify("Error reported edited successfully";Success);;Navigate(TroubleShooting)

PowerAutomate Flows :

RPA_Monitoring_Tool_ScheduledJobs:

This PowerAutomate flow connect with the uiPath orchestrator app via client secret Http Request and store the parsed information in the RPA_ScheduledProcesses Sharepoint List.

Note: The is a cleanup RPA_Monitoring_Tool_ScheduledJobs in order to keep the sharepoint data lower than 2000 rows, apply that to all the other flows that manages data in this app or consider to only filter for the last month data



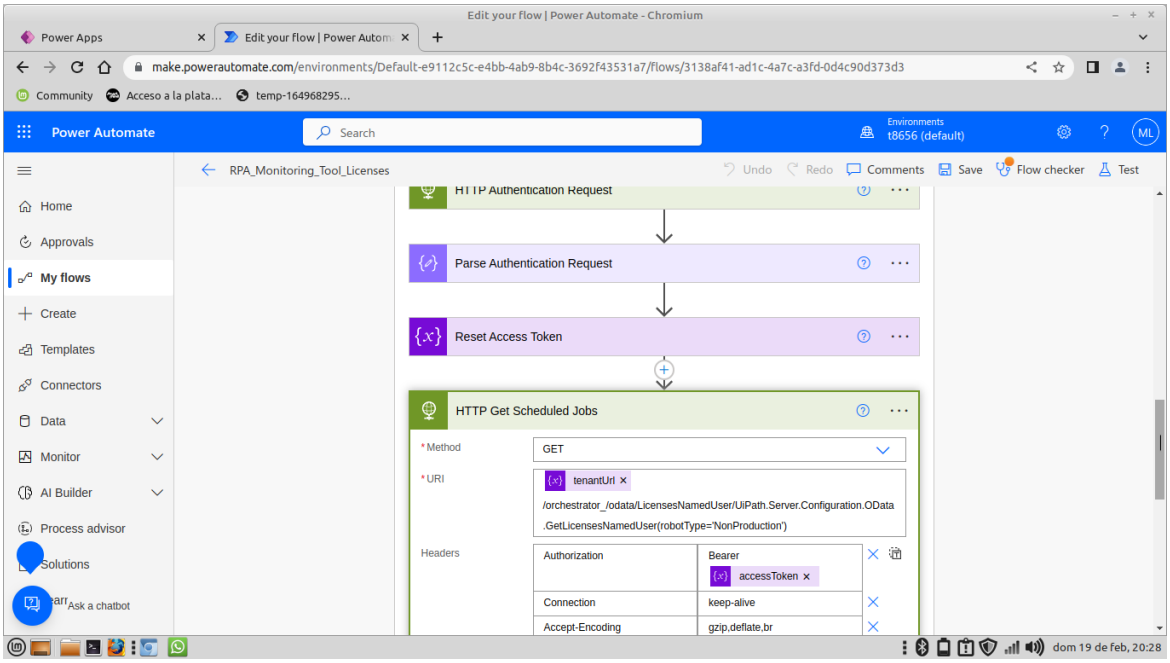
Picture 9: RPA_Monitoring_Tool_ScheduledJobs

The Http request swagger url is the following one :_

'tenantUrl'/orchestrator_/odata/ProcessSchedules

RPA_Monitoring_Tool_Licenses:

This PowerAutomate flow connect with the uiPath orchestrator app via client secret Http Request and store the parsed information in the RPA_ErrorLogs Sharepoint List.



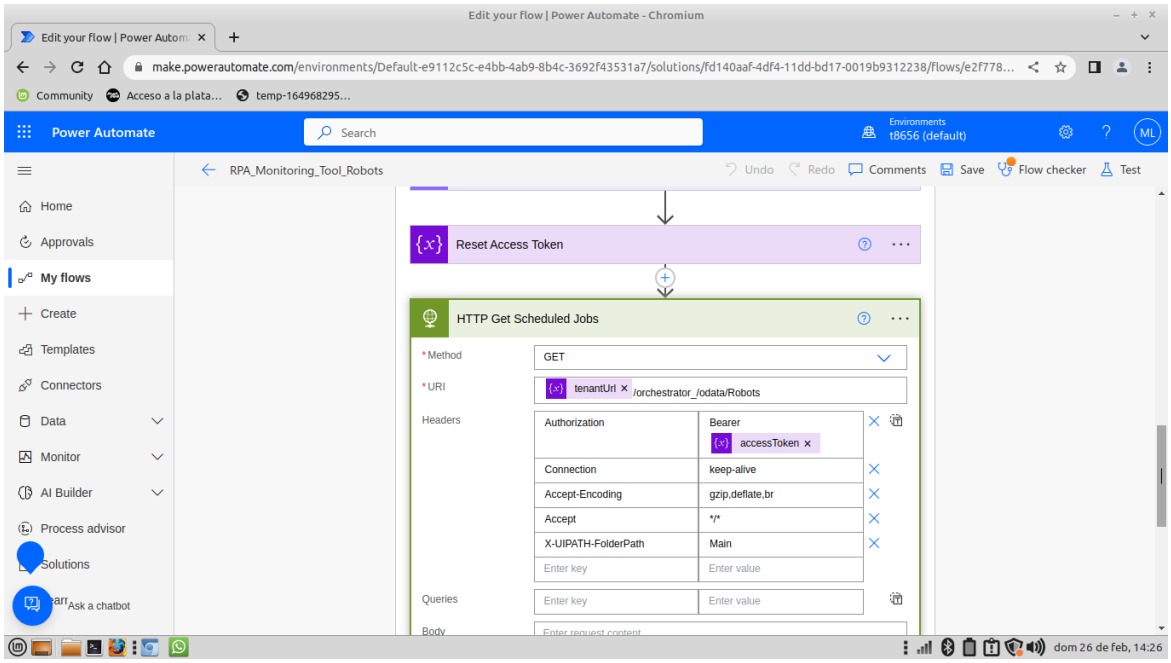
Picture 10: RPA_Monitoring_Tool_Licenses

The Http request swagger url is the following one :_

'tenantUrl'/orchestrator_/odata/LicensesNamedUser/
UiPath.Server.Configuration.OData.GetLicensesNamedUser(robotType='NonPro
duction')

RPA_Monitoring_Tool_Robots:

This PowerAutomate flow connect with the uiPath orchestrator app via client secret Http Request and store the parsed information in the RPA_ErrorLogs Sharepoint List.



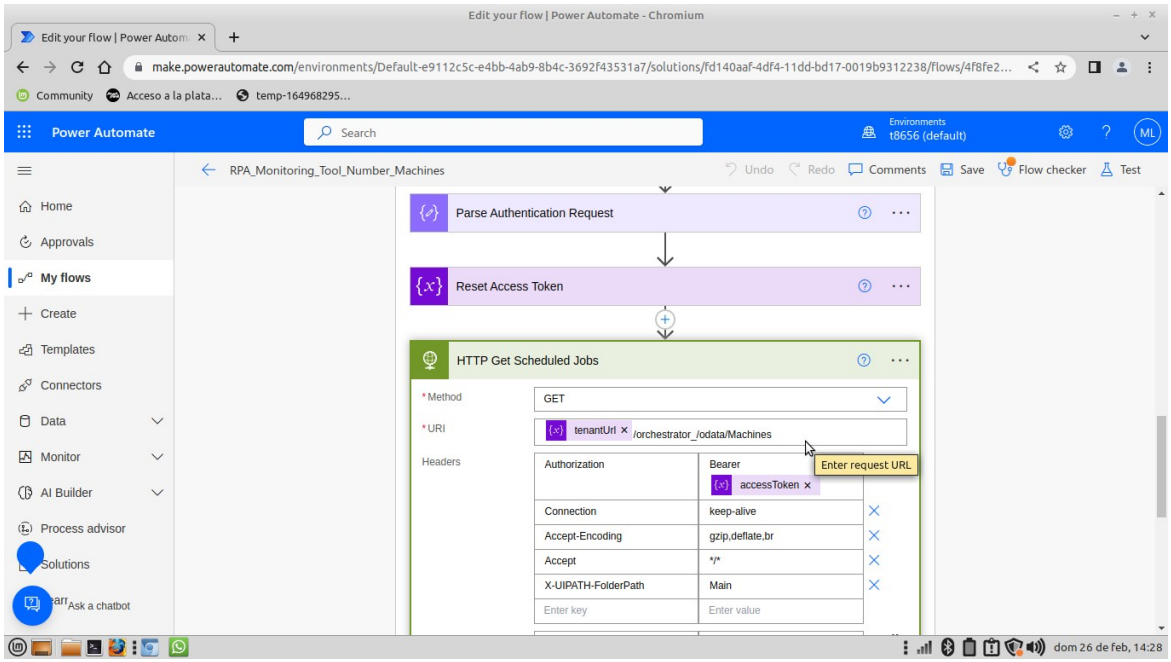
Picture 11: RPA_Monitoring_Tool_Robots

The Http request swagger url is the following one :

'tenantUrl'/orchestrator_/odata/Robots

RPA_Monitoring_Tool_Number_Machines:

This PowerAutomate flow connect with the uiPath orchestrator app via client secret Http Request and store the parsed information in the RPA_ErrorLogs Sharepoint List.



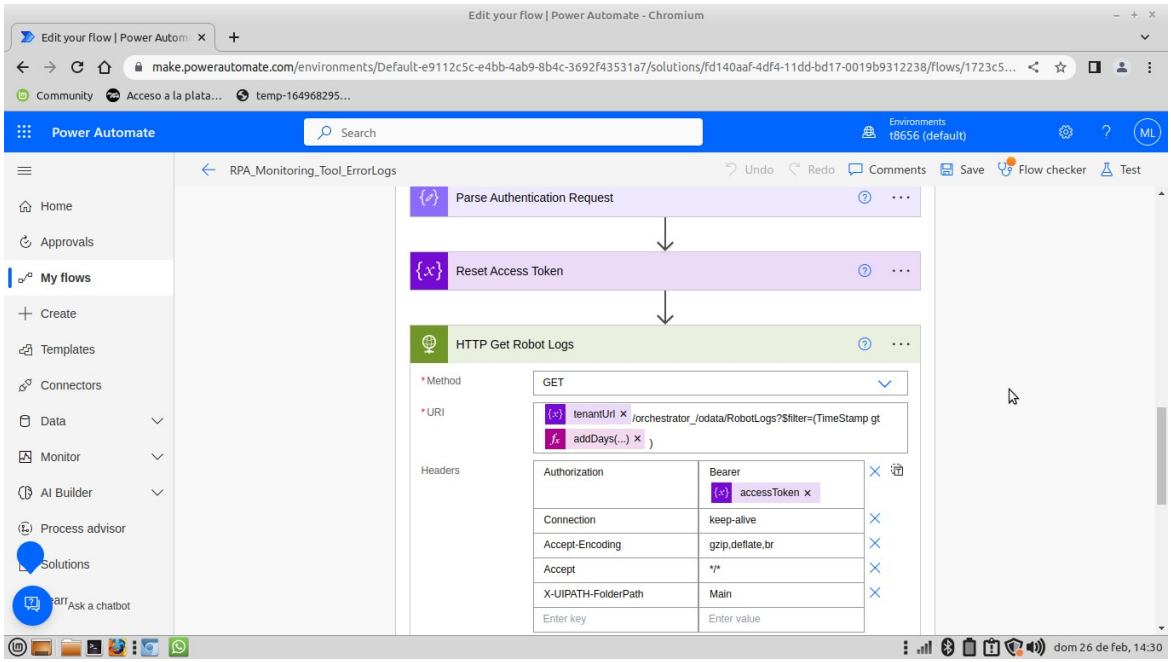
Picture 12: RPA_Monitoring_Tool_Number_Machines

The Http request swagger url is the following one :

'tenantUrl'/orchestrator_/odata/Machines

RPA_Monitoring_Tool_ErrorLogs:

This PowerAutomate flow connect with the uiPath orchestrator app via client secret Http Request and store the parsed information in the RPA_ErrorLogs Sharepoint List.



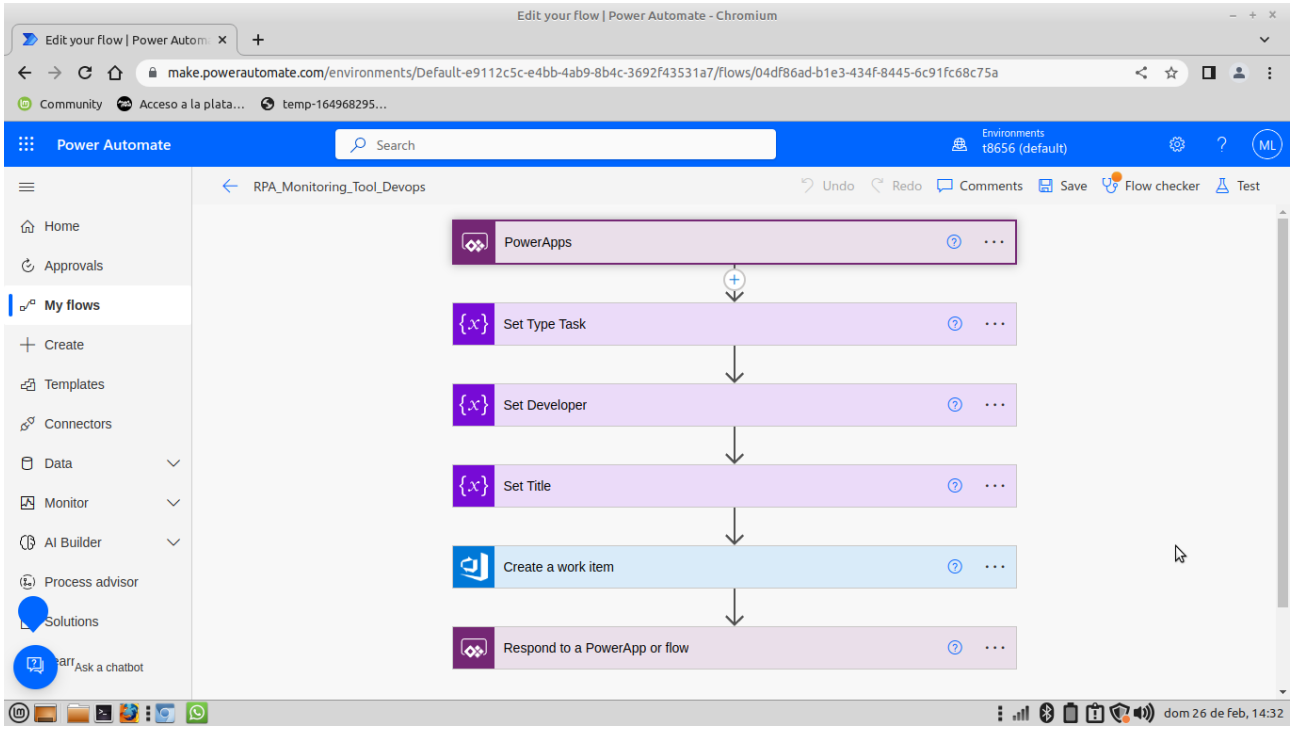
Picture : RPA_Monitoring_Tool_ErrorLogs

The Http request swagger url is the following one :

'tenantUrl'/orchestrator_/odata/RobotLogs?$filter=(TimeStamp gt addDays(body('Convert_time_zone'),-1))

RPA_Monitoring_Tool_Devops:

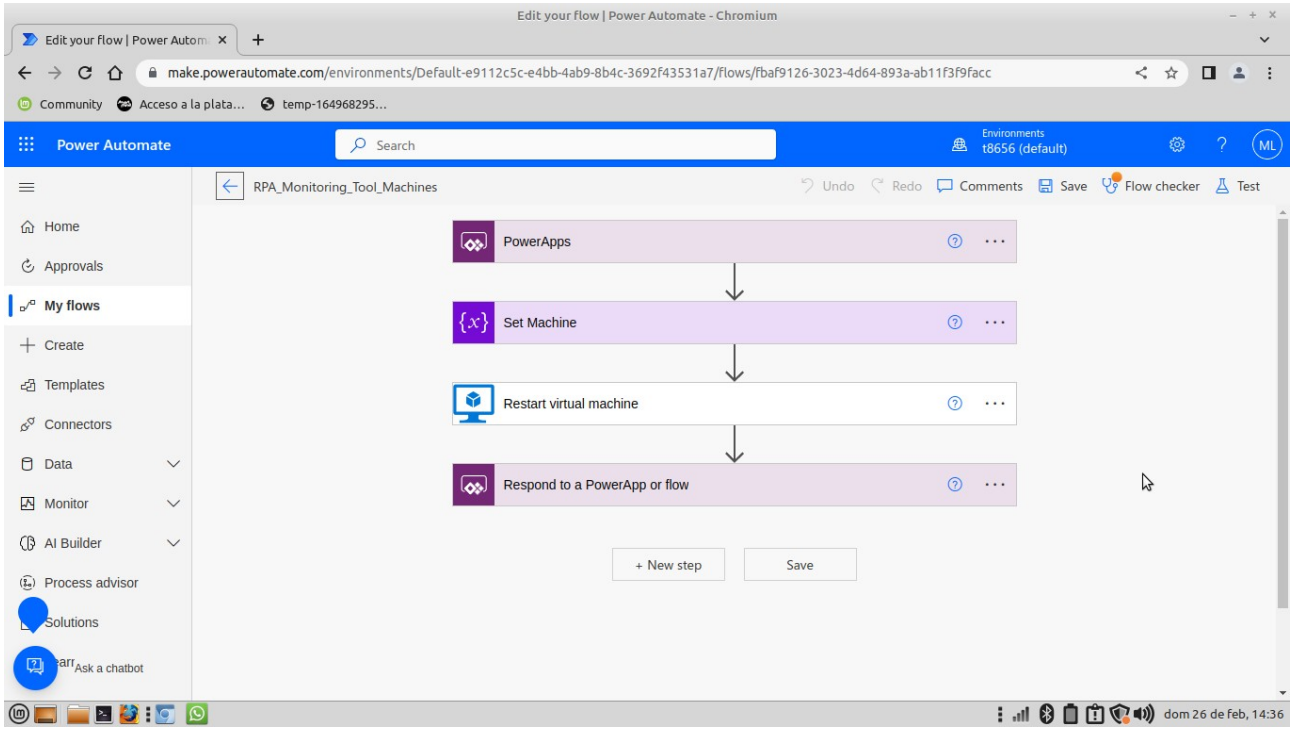
The PowerAutomate flow get the variables Type Taks , Developer and Title from the Powerapps and Create a new work item in the destination Azure Devops.



Picture 14: RPA_Monitoring_Tool_Devops

RPA_Monitoring_Tool_Machines:

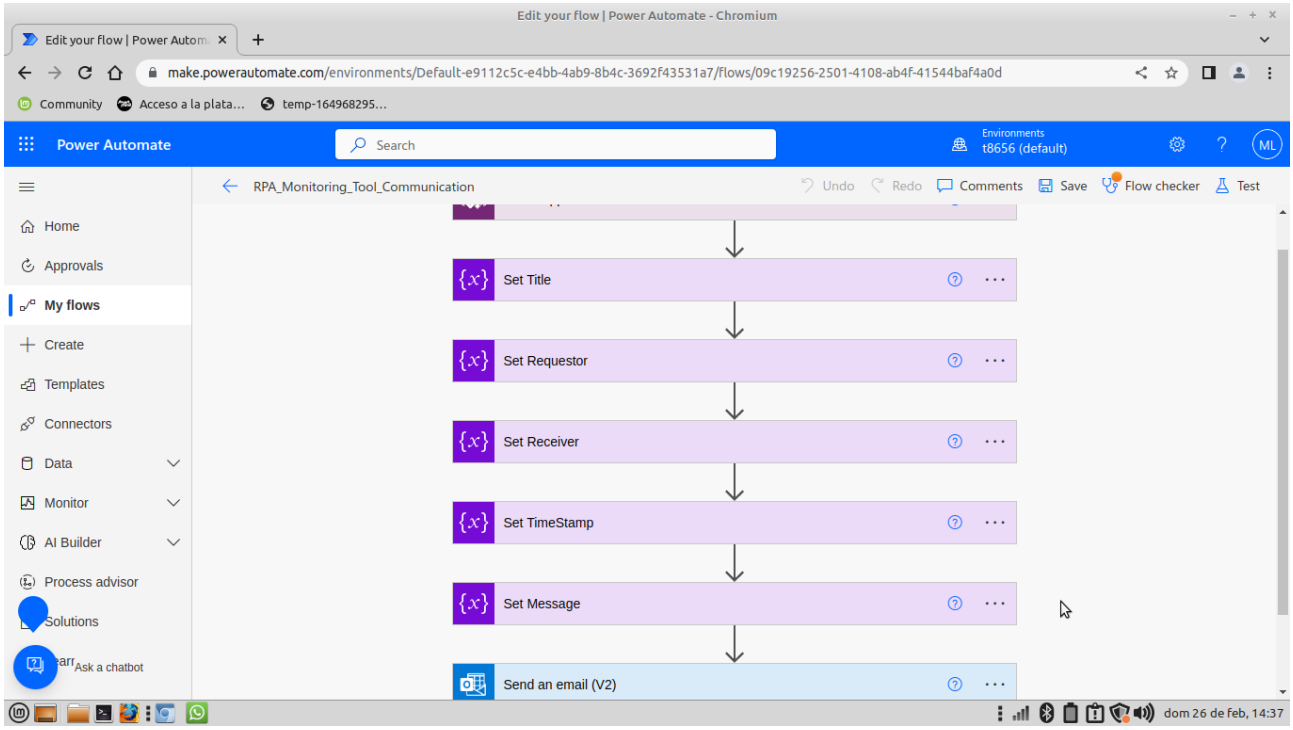
The PowerAutomate flow get the variables Machine from the Powerapps and restart the selected virtual machine.



Picture 15: RPA_Monitoring_Tool_Machines

RPA_Monitoring_Tool_Communication:

The PowerAutomate flow get the variables from the Powerapps and send an email to the destinatary.



Picture 16: RPA_Monitoring_Tool_Communication