Table of Contents

INTRODUCTION

This documentation is intended to be a comprehensive guide for anyone responsible for maintaining/troubleshooting or improving the quality inspection app in the future. This documentation is divided into two sections. The first section covers all the frontend elements: these are all the screens and the underlying codes for the screens in powerapp. The second section covers the backend components including the flows used in Microsoft Flow, data sources and data types. Refer back to the table of content while troubleshooting on any specific screen. Without any further ado, Let's get started with the frontend first.

FRONTEND OF POWERAPP

SCREEN 1: WELCOME SCREEN

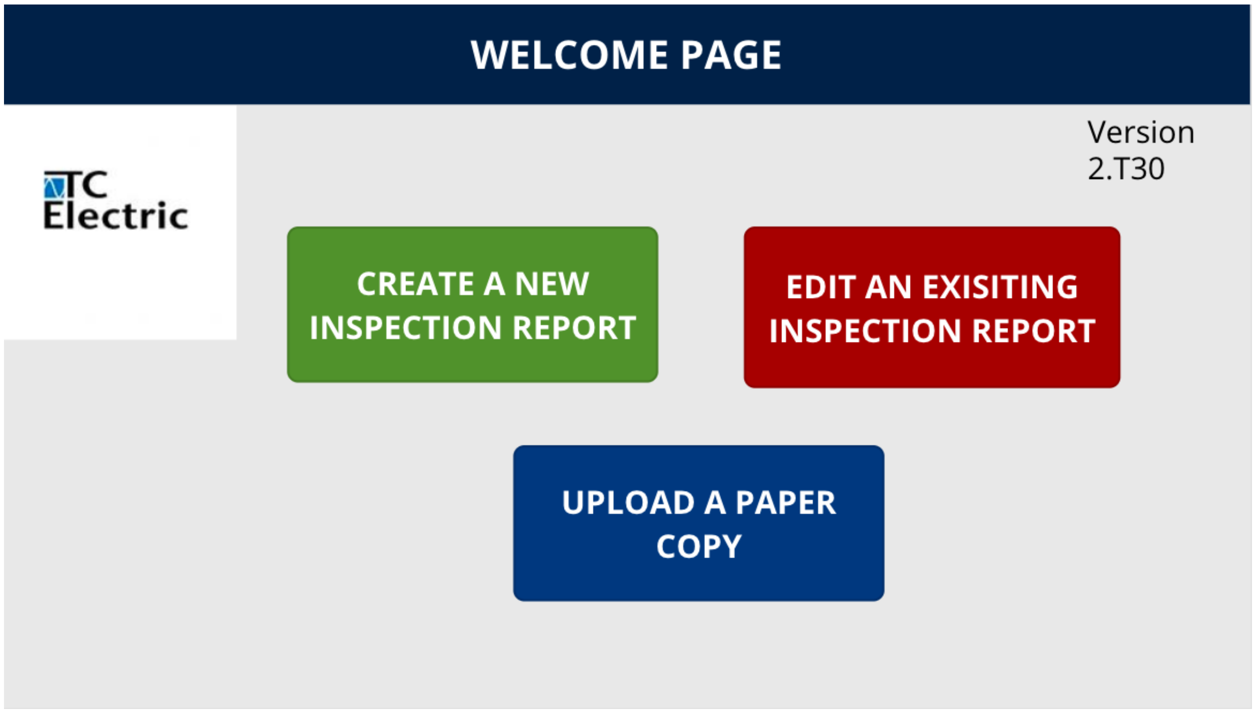

WELCOME PAGE
Version 2.T30
TC Electric
CREATE A NEW INSPECTION REPORT
EDIT AN EXISITING INSPECTION REPORT
UPLOAD A PAPER COPY

Image: screen 1 welcome screen

The top left corner contains the version number. The 'T' stands for tablet mode. You can find older versions of the app in the storage that was designed for a mobile setup. In this documentation however we will cover only the tablet mode of the app. It is recommended that

once any change is made to the app, the developer changes the version number on the top just to keep track of the different versions.
This screen contains three buttons: create a new report, edit an existing report and upload a new report. These three are the primary functionalities of the app. The purposes are self explanatory. However, I will go over the underlying codes for each button.

Create a new report:
Navigate(JobScreen4NewReport, ScreenTransition.Fade);
Reset(job)

The code simply tells us to navigate to a different screen when the button is pressed. It resets the job number. As you will cover the screen titled JobScreen4NewReport, you will see that there is a variable called job and this button makes sure that the variable job is refreshed every time we create a new report.

Edit an existing report:
Navigate(JobScreenEditExisting, ScreenTransition.Fade);
Reset(job_2);
Reset(Checkbox1)

It navigates to a different screen, resets the job variable and also resets a variable called checkbox1. These two variables are specific to the screen where the button is navigating us to.

Upload a paper copy:
Navigate(JobScreen4Upload, ScreenTransition.Fade);
Reset(job_1)

Navigates to a new screen and resets the job variable in that screen.

SCREEN 2: JobScreen4NewReport

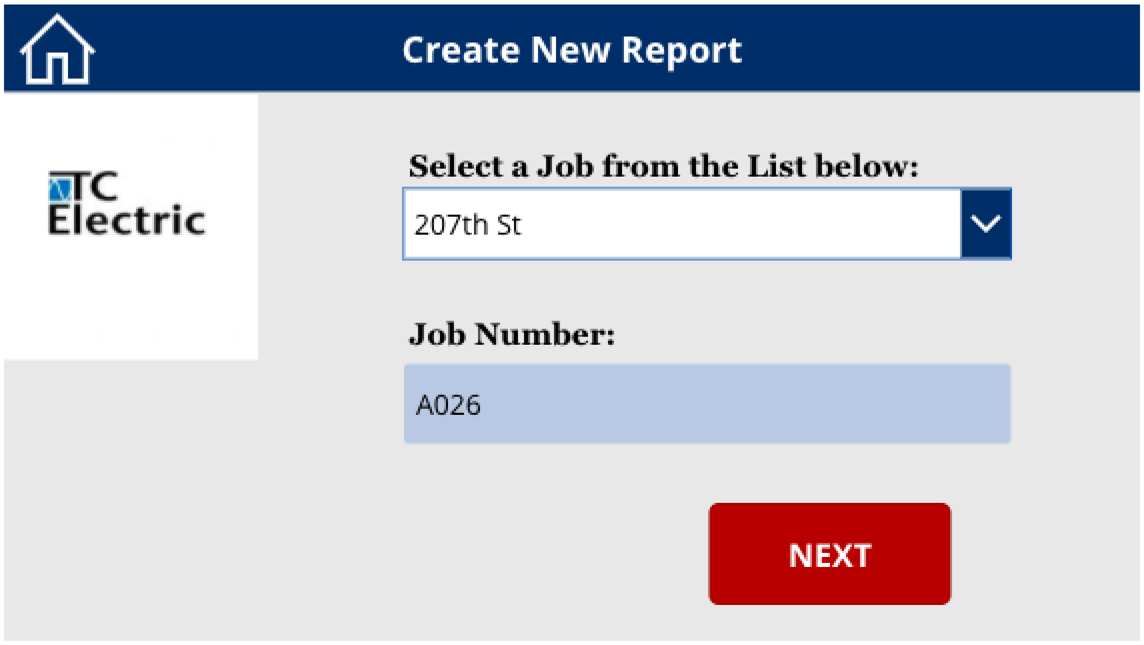
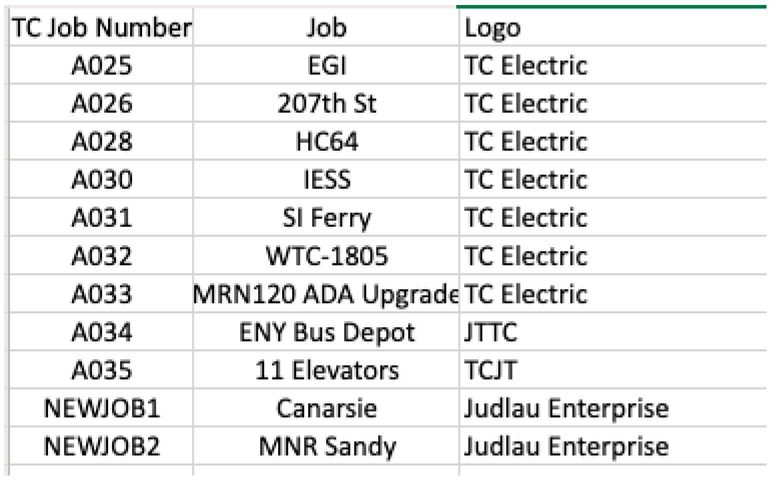


Image: Screen 2

There are two elements here: job name and job number. Once a job name is selected from the first drop down menu, the job number is automatically matched based on the first selection.

The first drop down menu is pre populated with a set of job names. The job names came from an excel file which also included the job number with respect to each job name. The excel file I used is shown below:



TC Job Number	Job	Logo
A025	EGI	TC Electric
A026	207th St	TC Electric
A028	HC64	TC Electric
A030	IESS	TC Electric
A031	SI Ferry	TC Electric
A032	WTC-1805	TC Electric
A033	MRN120 ADA Upgrade	TC Electric
A034	ENY Bus Depot	JTTC
A035	11 Elevators	TCJT
NEWJOB1	Canarsie	Judlau Enterprise
NEWJOB2	MNR Sandy	Judlau Enterprise

The first dropdown will contain all the values from column 2 in a list and the second column will match the correct job number from the first column. This matching process is done by a series of If-Else conditional statements (I couldn't find any hashmap or dictionary style data structure for powerapp and thus the use of if-else).

If you click on the first drop down, this is what it contains under the hood:
["EGI","207th St","HC64","IESS","SI Ferry","WTC-1805","MRN120 ADA Upgrade","ENY Bus Depot","11 Elevators","Canarsie","MNR Sandy"]

If you click on the second label, this is what it contains:

If(job.Selected.Value="EGI","A025",job.Selected.Value="207th St","A026",job.Selected.Value="HC64","A028",job.Selected.Value="IESS","A030",job.Selected.Value="SI Ferry","A031",job.Selected.Value="WTC-1805","A032",job.Selected.Value="MRN120 ADA Upgrade","A033",job.Selected.Value="ENY Bus Depot","A034",job.Selected.Value="11 Elevators","A035",job.Selected.Value="Canarsie","NEWJOB1",job.Selected.Value="MNR Sandy","NEWJOB2")

Note that these data in the excel file can change. So it will be extremely annoying to keep changing these if-else statements everytime a change occurs in the excel data. To automate this process, I wrote a Python script which basically takes in the Excel file as an input and generates the codes for the powerapp in a text file. Then you will just copy and paste that code into the powerapp. I am attaching my Python code below, followed by its output in the text file:

<u>Python Code for automating powerapp code:</u>

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 16 15:19:06 2020

TC ELECTRIC LLC

@author: Reaz
"""

import pandas as pd
import os,re,sys



class power_app_code(object):

    def __init__(self,code_file,excel_file):
        self.job_number=[]
        self.job =[]
        self.logo=[]
        self.logo_map={'TC Electric':100,'Judlau Enterprise':200,'JTTC':300,'TCJT':400}
        #self.logo_map is a hashmap for storing the integer value of each logo type

        self.excel_file=excel_file
        self.code_file=code_file


    def writer (self,filename,content):

        with open(filename,"w") as file:
            file.write(content)

#       file=open(self,filename,'w')
#       file.write(content)
#       file.close()

    def reader(self,filename):

        file=open(filename,"r")
        content=file.read()
        file.close()
        return content


    def appender (self,filename,content):

        file=open(filename,'a')
        file.write(content)
        file.close()


    def create_list_from_excel(self):


        df=pd.read_excel(self.excel_file)

        columns=list(df.columns)
        self.job_number=list(df[columns[1]])
        self.job=list(df[columns[0]])
        self.logo=list(df[columns[2]])

    def stringify_job_number(self):

        if len(self.job)==0 or len(self.job_number) or len(self.logo)==0:
            self.create_list_from_excel()
        self.writer(self.code_file,'[')
#        self.appender(self.code_file,'" "')
#        self.appender(self.code_file,',')
        for x in self.job_number:
            self.appender(code_file,'"')
            self.appender(code_file,x)
            self.appender(code_file,'"')
            self.appender(code_file,",")

        self.appender(code_file,']')
        self.appender(code_file,"\n")
        self.appender(code_file,"\n")



    def stringify_job(self,which_job):

        ## which job is going to vary with
        # create, edit and upload
        #create=job
        #edit=job_2
        #upload=job_1

        if len(self.job)==0 or len(self.job_number) or len(self.logo)==0:
            self.create_list_from_excel()

        job_str=[]
        job_str.append(f'If({which_job}.Selected.Value="{self.job_number[0]}","{self.job[0]}",')

self.appender(self.code_file,f'If({which_job}.Selected.Value="{self.job_number[0]}","{self.job[0]}",')

        for i in range(1,len(self.job)):
            job_str.append(f'{which_job}.Selected.Value="{self.job_number[i]}","{self.job[i]}"')


        for i in range(1,len(self.job)-1):
            self.appender(code_file,job_str[i])
            self.appender(code_file,',')

        self.appender(code_file,job_str[len(job_str)-1])
        self.appender(code_file,')')
        self.appender(code_file,"\n")
        self.appender(code_file,"\n")



    def stringify_logo(self,which_job):
        ## which job is going to vary with
        # create, edit and upload
        #create=job
        #edit=job_2
        #upload=job_1

        if len(self.job)==0 or len(self.job_number) or len(self.logo)==0:
            self.create_list_from_excel()

        logo_str=[]

self.appender(self.code_file,f'If({which_job}.Selected.Value="{self.job_number[0]}","{self.logo_map[sel
f.logo[0]]}",')

        for i in range(1,len(self.job)):

logo_str.append(f'{which_job}.Selected.Value="{self.job_number[i]}","{self.logo_map[self.logo[i]]}"')

        for i in range(1,len(self.job)-1):
            self.appender(code_file,logo_str[i])
            self.appender(code_file,',')

        self.appender(code_file,logo_str[len(logo_str)-1])
        self.appender(code_file,')')
        self.appender(code_file,"\n")
        self.appender(code_file,"\n")




excel_file='TC Job Listing_second.xlsx'
#excel_file='experimental.xlsx'
code_file='power_app_code.txt'



a=power_app_code(code_file,excel_file)
a.stringify_job_number()

#create new report
a.stringify_job('job')
a.stringify_logo('job')

#edit existing report
a.stringify_job('job_2')
a.stringify_logo('job_2')

#upload report
a.stringify_job('job_1')
```

```
a.stringify_logo('job_1')
```

Run this code on any Python interpreter on your machine and it will generate the following output in the text file called power_app_code.txt. I am numbering these below to refer back in the later sections of this documentation:

1. ***["EGI","207th St","HC64","IESS","SI Ferry","WTC-1805","MRN120 ADA Upgrade","ENY Bus Depot","11 Elevators","Canarsie","MNR Sandy",]***

2. ***If(job.Selected.Value="EGI","A025",job.Selected.Value="207th St","A026",job.Selected.Value="HC64","A028",job.Selected.Value="IESS","A030",job.Selected.Value="SI Ferry","A031",job.Selected.Value="WTC-1805","A032",job.Selected.Value="MRN120 ADA Upgrade","A033",job.Selected.Value="ENY Bus Depot","A034",job.Selected.Value="11 Elevators","A035",job.Selected.Value="Canarsie","NEWJOB1",job.Selected.Value="MNR Sandy","NEWJOB2")***

3. ***If(job.Selected.Value="EGI","100",job.Selected.Value="HC64","100",job.Selected.Value="IESS","100",job.Selected.Value="SI Ferry","100",job.Selected.Value="WTC-1805","100",job.Selected.Value="MRN120 ADA Upgrade","100",job.Selected.Value="ENY Bus Depot","300",job.Selected.Value="11 Elevators","400",job.Selected.Value="Canarsie","200",job.Selected.Value="MNR Sandy","200",job.Selected.Value="MNR Sandy","200")***

4. ***If(job_2.Selected.Value="EGI","A025",job_2.Selected.Value="207th St","A026",job_2.Selected.Value="HC64","A028",job_2.Selected.Value="IESS","A030",job_2.Selected.Value="SI Ferry","A031",job_2.Selected.Value="WTC-1805","A032",job_2.Selected.Value="MRN120 ADA Upgrade","A033",job_2.Selected.Value="ENY Bus Depot","A034",job_2.Selected.Value="11 Elevators","A035",job_2.Selected.Value="Canarsie","NEWJOB1",job_2.Selected.Value="MNR Sandy","NEWJOB2")***

5. ***If(job_2.Selected.Value="EGI","100",job_2.Selected.Value="HC64","100",job_2.Selected.Value="IESS","100",job_2.Selected.Value="SI***

Ferry","100",job_2.Selected.Value="WTC-1805","100",job_2.Selected.Value="MRN120 ADA Upgrade","100",job_2.Selected.Value="ENY Bus Depot","300",job_2.Selected.Value="11 Elevators","400",job_2.Selected.Value="Canarsie","200",job_2.Selected.Value="MNR Sandy","200",job_2.Selected.Value="MNR Sandy","200")

6. ***If(job_1.Selected.Value="EGI","A025",job_1.Selected.Value="207th St","A026",job_1.Selected.Value="HC64","A028",job_1.Selected.Value="IESS","A030",job_1.Selected.Value="SI Ferry","A031",job_1.Selected.Value="WTC-1805","A032",job_1.Selected.Value="MRN120 ADA Upgrade","A033",job_1.Selected.Value="ENY Bus Depot","A034",job_1.Selected.Value="11 Elevators","A035",job_1.Selected.Value="Canarsie","NEWJOB1",job_1.Selected.Value="MNR Sandy","NEWJOB2")***

7. ***If(job_1.Selected.Value="EGI","100",job_1.Selected.Value="HC64","100",job_1.Selected.Value="IESS","100",job_1.Selected.Value="SI Ferry","100",job_1.Selected.Value="WTC-1805","100",job_1.Selected.Value="MRN120 ADA Upgrade","100",job_1.Selected.Value="ENY Bus Depot","300",job_1.Selected.Value="11 Elevators","400",job_1.Selected.Value="Canarsie","200",job_1.Selected.Value="MNR Sandy","200",job_1.Selected.Value="MNR Sandy","200")***

1) And 2) gives us the list of the jobs and the code to associate each job name with its corresponding job number.

<u>HIDDEN BUTTON FOR LOGO:</u>

Note that in addition to matching job name with job number, when a job number is selected it also matches that job with a specific logo. In the excel file the third column shows the arrangements of these logos. If you look at the code generated by the Python script- The third paragraph shows the code for matching job names with specific logo numbers. The logos are numbered as 100,200, 300 and 400. The schema for naming these is also defined in the Python code shown above:
self.logo_map={'TC Electric':100,'Judlau Enterprise':200,'JTTC':300,'TCJT':400}

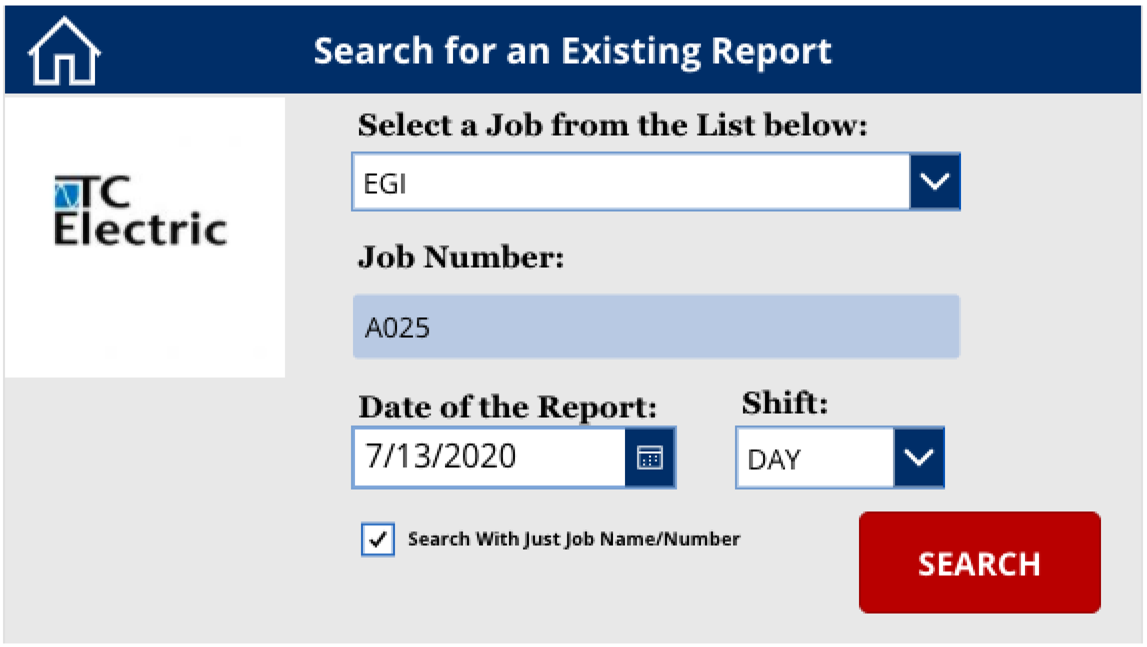
Note that the logo selection in the Powerapp is done via a hidden button. If you look at all the components under the screen you will see a hidden button called **logo_picker** containing the 4 logo numbers: 100,200,300 and 400. The "Default" value of this element contains the necessary code to map it with the job name. This code is numbered as (3) above. The actual images of the logos are in the OneDrive storage.The powerapp only provides the path in OneDrive to get these images. We will discuss this in more detail in the backend section of this document.

Now, let's look at the code for this **NEXT** button in red:

```
NewForm(EditForm1_1);
UpdateContext({WEATHER:MSNWeather.CurrentWeather("11417","Imperial")});

Navigate(
   EditScreenNewReport,
   ScreenTransition.Fade,
   {
      job: job.Selected.Value,
      job_number: job_number.Text,
      logo_path:logo_picker.Selected.Value,
      weather_info:WEATHER
   }
)
```

Explanation: it resets the EditForm in the next screen. An MSN weather API is connected to the app for getting up to date weather information. The number 11417 is my home zip code in Queens. Under Navigate, it collects information such as job, job_number, logo_path and weather info and passes these information onto the next screen. The next screen will then pass this information onto the backend for preparing a full report, as will be discussed later.

SCREEN 3: JobScreenEditExisting


Search for an Existing Report
TC Electric
Select a Job from the List below:
EGI
Job Number:
A025
Date of the Report:
7/13/2020
Shift:
DAY
Search With Just Job Name/Number
SEARCH

This will be the second screen had the user selected Edit Existing Report from the welcome screen. This screen is very similar to the screen defined in the previous section. It contains all the code for matching job name with number and with logo. The logo picker label is named "logo_picker_dos" on this screen. The PowerApp code generated via Python is used in this screen as well, but instead of job and job_number it will have job_2, job_number_2 etc.

There are 3 additional elements in this screen: date, shift and a checkbox. These are used to search an existing report in the database.

HELPER LABEL:

This is a hidden label. You can find it under the screen breakdown. This helper label combines the information such as job name/number, date and shift into a single string called helper. This string is what's used to find an existing report from the database. Let's look at the code under this helper label:

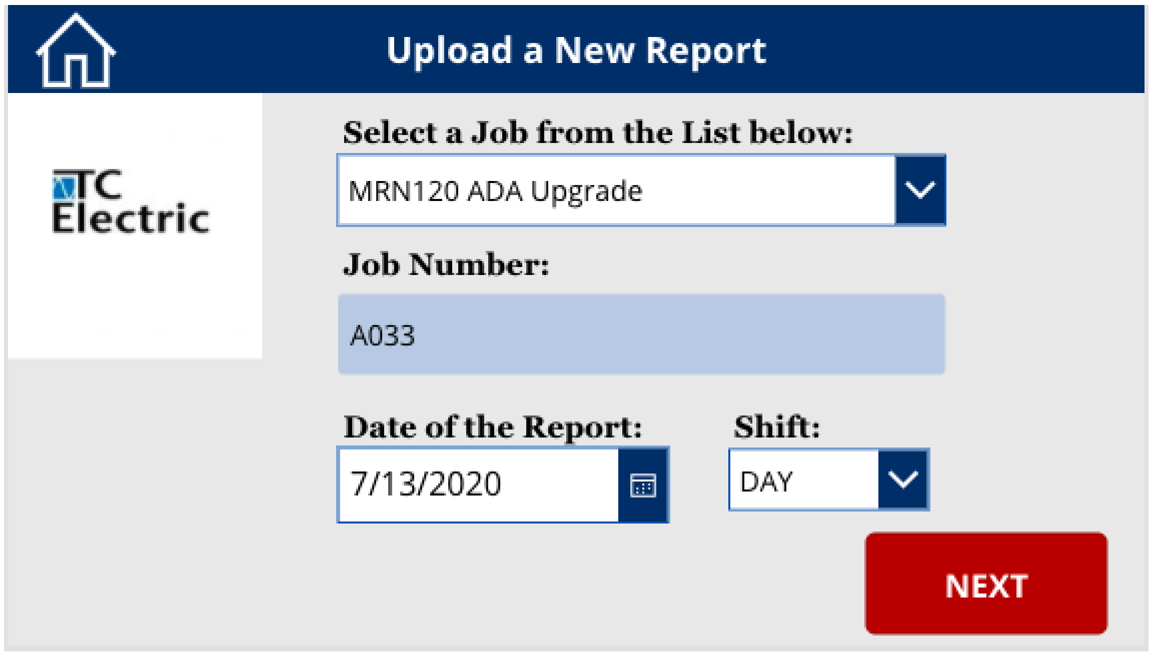
Concatenate(job_number_2.Text,"-",Text(DatePicker1_1.SelectedDate),Text(lookup_shift.Selected.Value))

Now, let's look at the code inside the SEARCH button:

If(Checkbox1.Value=false,Navigate(BrowseScreen,Fade,{helper:helper_label.Text});,Checkbox1.Value=true,Navigate(BrowseScreen,Fade,{helper:job_number_2.Text}))

It says that if the checkbox is selected then use just the job number to look up the existing report, otherwise create a concatenated string to look up. And the string is our helper label discussed earlier. Finally navigate to the next screen and pass the helper label for searching.

SCREEN 4: JobScreen4Upload



This will be the second screen for the user had he selected Upload a Paper Copy from the welcome screen. It is very similar to the two screens discussed earlier. Let's just look at the code for the NEXT button:

```
Navigate(
   UploadReportScreen,
   ScreenTransition.Fade,
   {
      job: job_1.Selected.Value,
      job_number: job_number_1.Text,
      date:Text(DatePicker1.SelectedDate,"[$-en-US]mm_dd_yy"),
      uploaded_report_name:Concatenate(job_number_1.Text,"-
",Text(DatePicker1.SelectedDate,"[$-en-
US]mm_dd_yy"),Text(shift_upload.Selected.Value),".html"),
      doc_name: Concatenate(job_number_1.Text,"-",Text(DatePicker1.SelectedDate,"[$-en-
US]mm_dd_yy"),Text(shift_upload.Selected.Value),".docx")
```

```
   }
);
Reset(upload_pic_button)
```

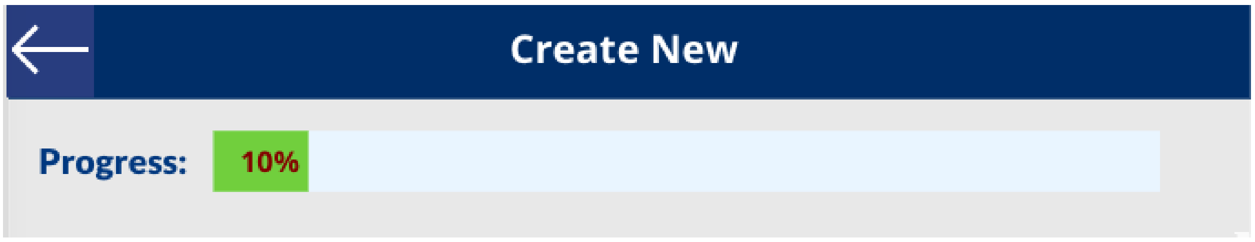
Explanation: It collects information such as job name and number, date and shift to create a name for the report. Notice that there are two variables: uploaded report name and doc_name. The former is for creating an HTML type document and the latter is for creating a Word document. Although the app supports both types, HTML is the one that gets sent to the user due to better image rendering capability. We will talk about that more in the later sections.

SCREEN 5: EditScreenNewReport

This is the screen where the user will enter all the necessary information to create a paper copy equivalent of the inspection report. All the fields for this screen are defined in an excel file and the PowerApp automatically generates the screen from those defined fields. The Excel file used to create these fields is called: Central_Form1.xlsx and the table is named Table2 (sorry for this generic name, Excel's online version won't let me name tables manually).

Most of the stuff in this screen is self explanatory. I will only go over the important elements and their codes in this section. Let's start with the progress bar up top.

PROGRESS BAR



You can find the progress bar labeled as "progress" under the screen. For the Text value, it has the following code:

```
//progress Text

// for progress text
```

```
If(And(cp1.Text="0",cp2.Text="0"),"0%",
And(cp1.Text="1",cp2.Text="0"),"10%",
And(cp1d.Text="1",cp2.Text="1",cp3.Text="0"),"40%",
And(cp1.Text="1",cp2.Text="1",cp3.Text="1",cp4.Text="0"),"80%",
And(cp1.Text="1",cp2.Text="1",cp3.Text="1",cp4.Text="1"),"100%")
```

Let's look at the 80 percent as an example (4th line), it says when cp1, cp2, cp3 are 1 but cp4 is 0, set the progress to 80 percent. You might be wondering what the hell are cp1, cp2 etc. The progress bar is divided into 5 parts. It will always start out with 10 percent as long as job and job numbers are filled. These two fields will be automatically filled as the user would have entered those info in the previous screen (Screen 2).

Then 30 percent is assigned to the rest of the Project Information fields. When all these fields are filled, the progress bar will be set to 40 percent. Now, cp1, cp2 are hidden labels that can either have a value of 0 or 1. Cp stands for checkpoint. You can find these under the screen. Cp1 is under datacard Contract No, Cp2 is under data card Completed by, Cp3 is under additional comment and cp4 is under date2. The progress bar text depends on the values of these checkpoints markers. The user doesn't need to see these checkpoints (we have the progress bar to inform the user), that's why the labels are hidden. I used these labels to make the code more readable. I highly recommend you take a look at all the individual codes for these checkpoint labels on your own. But let's just take a look at the logic for cp2 together:

```
If(Or(
   IsBlank(description),
   IsBlank(work_location),
   IsBlank(work_performed_by),
   IsBlank(items_installed),
   IsBlank(approved_by),
   IsBlank(spec_section),
   IsBlank(completed_by)

   ),

   "0",
   "1")
```

Explanation: So, the code is basically saying if any of the mentioned data cards such as description, work_location etc. is empty set the value for this label to be zero. The 'OR' operation will set the value to zero if one or more elements are empty. Finally, if everything is filled, set it to 1.

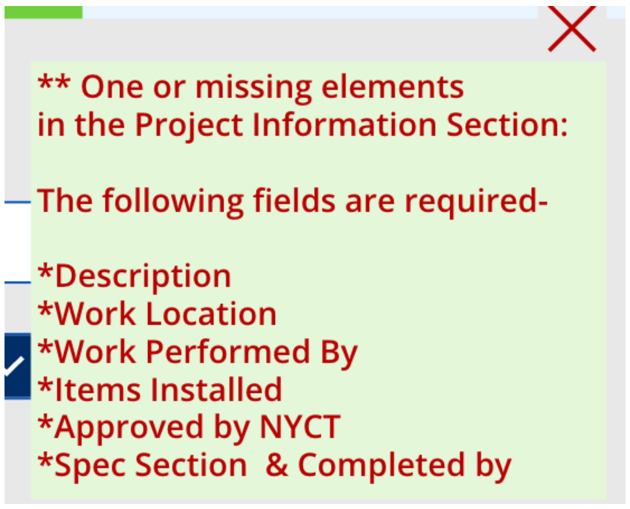
The progress bar has two important sub-elements: blue bar and green bar. Green bar is the one that moves along with progress. The blue bar is the container that holds the green bar inside it. The program basically adjusts the width of this green bar as new checkpoints are fulfilled. Let’s look at the code for the Width of this green bar:

```
//green bar width
// for green_bar width
If(And(cp1.Text="0",cp2.Text="0"),64,
And(cp1.Text="1",cp2.Text="0"),blue_bar_1.Width*0.1,
And(cp1.Text="1",cp2.Text="1",cp3.Text="0"),blue_bar_1.Width*0.4,
And(cp1.Text="1",cp2.Text="1",cp3.Text="1",cp4.Text="0"),blue_bar_1.Width*0.8,
And(cp1.Text="1",cp2.Text="1",cp3.Text="1",cp4.Text="1"),blue_bar_1.Width*1)
```

As you can see depending on the values of the checkpoints, the width is adjusted from 10 percent of the blue bar’s width to 100 percent of the blue bar’s width.

Next, let’s take a look at the feedback popup feature which also relies on the checkpoints.

FEEDBACK POPUP



** One or missing elements
in the Project Information Section:

The following fields are required-

*Description
*Work Location
*Work Performed By
*Items Installed
*Approved by NYCT
*Spec Section & Completed by

This popup window prevents the user from going to the next screen when they hit NEXT prematurely and tells explicitly which fields must be taken care of before moving on. Let’s take a look at the code for the Text value of this feedback popup. You can find it directly under the screen titled “feedback”.

```
If(
progress.Text="10%",
"** One or missing elements
in the Project Information Section:

The following fields are required-

*Description
*Work Location
*Work Performed By
*Items Installed
*Approved by NYCT
*Spec Section  & Completed by"


,progress.Text="40%",
"** One or missing elements

in the Inspection Plan Section:
Make sure you have answered all the questions

Valid Answers are: Yes, No or N/A "

,progress.Text="80%",
"** One or missing elements

in the Sign Section:
The following items are required:

*QM/QS
*Date (QM)
*Date (CCM)"
)
```

As you can see the feedback is reliant on the progress bar for its actions. There are if-else cases to display certain messages when the progress bar is at 10, 40 and 80 percent.

This feedback popup stays invisible when the screen loads. If you look at the InVisible code for the screen there are two variables: popup and cancel. Popup is the visibility status of the feedback window. If you look at the screenshot above, on the top right corner there is a red cross, that’s the cancel button. Once clicked the feedback popup disappears. Let’s look at the code for OnSelect inside the cancel button:

```
UpdateContext({popup:false,cancel:false})
```

Once again when popup is false, the feedback window is invisible. When the cancel is false, the little red cross on top is invisible. Cancel is the visibility status variable for the red cross sign.

NEXT BUTTON

Let’s look at the code first, this is probably the single most important code for this screen:

```
Set(uploaded_name1,Concatenate(job_number,"-",Text(qm_date.SelectedDate,"[$-en-US]mm_dd_yy"),Text(qm_shift.Selected.Value),"upload1"));
Set(uploaded_name2,Concatenate(job_number,"-",Text(qm_date.SelectedDate,"[$-en-US]mm_dd_yy"),Text(qm_shift.Selected.Value),"upload2"));

Set(uploaded_content1,If(Text(u1_image.Image)="appres://resources/SampleImage","No Image Uploaded",Substitute(JSON(u1_image.Image,JSONFormat.IncludeBinaryData),"""","")));
Set(uploaded_content1_compressed,If(Text(u1_image.Image)="appres://resources/SampleImage","No Image Uploaded",Substitute(uploaded_content1,Concatenate(First(Split(uploaded_content1,",")).Result,","),"")));

Set(uploaded_content2,If(Text(u2_image.Image)="appres://resources/SampleImage","No Image Uploaded",Substitute(JSON(u2_image.Image,JSONFormat.IncludeBinaryData),"""","")));
Set(uploaded_content2_compressed,If(Text(u2_image.Image)="appres://resources/SampleImage","No Image Uploaded",Substitute(uploaded_content2,Concatenate(First(Split(uploaded_content2,",")).Result,","),"")));
Set(image1_decision,If(Text(u1_image.Image)="appres://resources/SampleImage","NO","Y
ES"));
Set(image2_decision,If(Text(u2_image.Image)="appres://resources/SampleImage","NO","Y
ES"));
```

Explanation: It's a long one, so let's take a break here and explain what's going on. These codes in green are responsible for collecting the data from the uploaded images for question 1 and 2. The first two lines set the names for image1 and image2. The naming schemes are important because as we are storing these images in the backend, we want to be able to associate the images with their respective reports so that when someone wants to edit an old report, we can render the images from storage and put it with the rest of the report (we will talk more about it in the backend section).

Now the next two lines collect the actual content of the image. It says that if the image content is Sample Image which is powerapp's default for no image, set the content to "no image uploaded"; otherwise get the content of the image. Getting the content part may not seem so obvious in the code. After several trial and error I found that the best way to get image content and pass it on to the backend is to convert it into a base64 string and then trim the base64 string to get rid of the initial formatting description and just retain the raw content of the image. And this trimming is what the Substitute function does. The trimmed up content is named as uploaded_content1_compressed.

The last part is the decision maker, it keeps track of whether an image was uploaded or not. I kept it for troubleshooting, primarily.

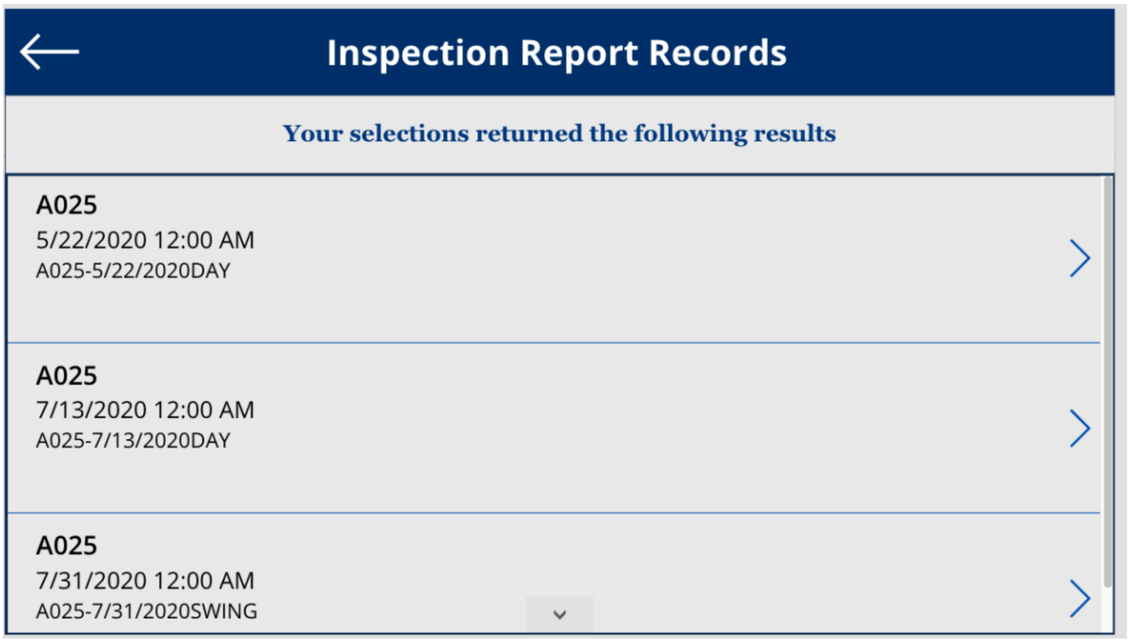
```
If(progress.Text="100%",
ClearCollect(
   coltest3,
   {
      contract_no: contract_no.Text,
      description: description.Text,
      work_location: work_location.Text,
      work_performed_by: work_performed_by.Text,
      items_installed: items_installed.Text,
      approved_by: approved_by.Text,
      spec_section: spec_section.Text,
      completed_by: completed_by.Text,
      a1: a1.Selected.Value,
      c1: c1.Text,
	a2: a2.Selected.Value,
	c2: c2.Text,
	a3: a3.Selected.Value,
	c3: c3.Text,
	a4: a4.Selected.Value,
	c4: c4.Text,
	a5: a5.Selected.Value,
	c5: c5.Text,
	a6: a6.Selected.Value,
	c6: c6.Text,
	a7: a7.Selected.Value,
	c7: c7.Text,
	a8: a8.Selected.Value,
	c8: c8.Text,
	a9: a9.Selected.Value,
	c9: c9.Text,
	a10: a10.Selected.Value,
	c10: c10.Text,
	a11: a11.Selected.Value,
	c11: c11.Text,
	a12: a12.Selected.Value,
	c12: c12.Text,
	qm_sign: qm_sign.Text,
	qm_date: qm_date.SelectedDate,
	qm_shift: qm_shift.Selected.Value,
	ccm_sign: ccm_sign.Text,
	ccm_date: ccm_date.SelectedDate,
	ccm_shift: ccm_shift_wrong.Selected.Value,
	additional_comment: additional_comment.Text,
	job_flow: job_name.Text,
	job_number_flow: contract_no.Text,
	logo_path:logo_path,
	file_name:Concatenate(job_number,"-",Text(qm_date.SelectedDate,"[$-en-
US]mm_dd_yy"),Text(qm_shift.Selected.Value),".html"),
	pdf_name:Concatenate(job_number,"-",Text(qm_date.SelectedDate,"[$-en-
US]mm_dd_yy"),Text(qm_shift.Selected.Value),".pdf"),
	doc_name:Concatenate(job_number,"-",Text(qm_date.SelectedDate,"[$-en-
US]mm_dd_yy"),Text(qm_shift.Selected.Value),".doc"),
	weather:Text(weather_info.responses.weather.current.cap),
        u1:If(Text(u1_image.Image)="appres://resources/SampleImage","No Image
Uploaded",Substitute(JSON(u1_image.Image,JSONFormat.IncludeBinaryData),"""","")),

        u2:If(Text(u2_image.Image)="appres://resources/SampleImage","No Image
Uploaded",Substitute(JSON(u2_image.Image,JSONFormat.IncludeBinaryData),"""",""))
    }
);
'text_flow_html_only-2'.Run(JSON(coltest3));
Navigate(FinishScreen4NewForm,Fade,{filename:Concatenate(job_number,"-
",Text(qm_date.SelectedDate,"[$-en-
US]mm_dd_yy"),Text(qm_shift.Selected.Value),".html")});
NewReportStoreImage.Run(uploaded_name1,uploaded_content1_compressed,uploaded_name
2,uploaded_content2_compressed,image1_decision,image2_decision);

//Navigate(WelcomeScreen);

,progress.Text <> "100%",UpdateContext({popup:true,cancel:true}));
```

Explanation: Ok so there is a lot going on in this section. We start off with collecting all the information on the screen and putting them into certain variables. This aggregation of data is called a "Collection" in powerapp. In the backend this data type is easily converted into a JSON. Notice that the collection is prompted to happen only if the progress bar is at a 100 percent. At the very end of the code, you will see that if the progress is not 100 percent (< > - this is the not symbol) then it prompts the feedback window to popup, we talked about in the previous section.

So, once all the data collection happens, the button will trigger two flows: tex_flow_html_only-2 and NewReportStoreImage. The textflow html flow passes all the data from the app to the backend to create an HTML version of the inspection report. The NewReportStoreImage collects the uploaded images and saves them separately in OneDrive to be accessed later. We will talk about these two flows in detail in the backend section.

Screen6: BrowseScreen

This screen shows users the search result for existing reports. There are two possibilities here: either the search parameters returned some files that matched the parameters or no the search found no matching file. Both outputs are shown below:

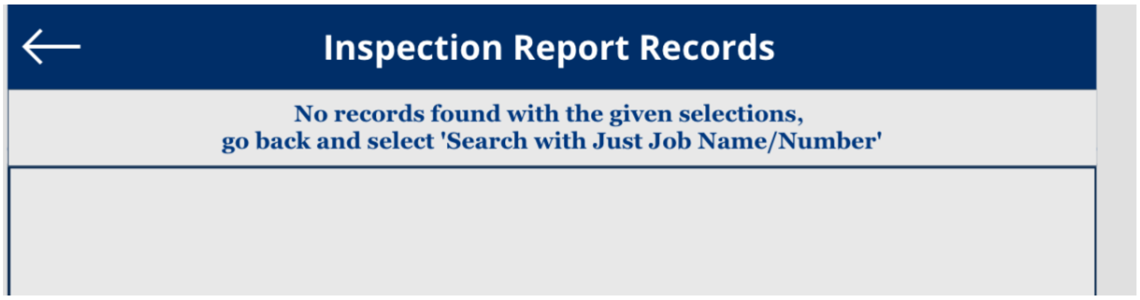

Inspection Report Records

Your selections returned the following results

A025
5/22/2020 12:00 AM
A025-5/22/2020DAY

A025
7/13/2020 12:00 AM
A025-7/13/2020DAY

A025
7/31/2020 12:00 AM
A025-7/31/2020SWING

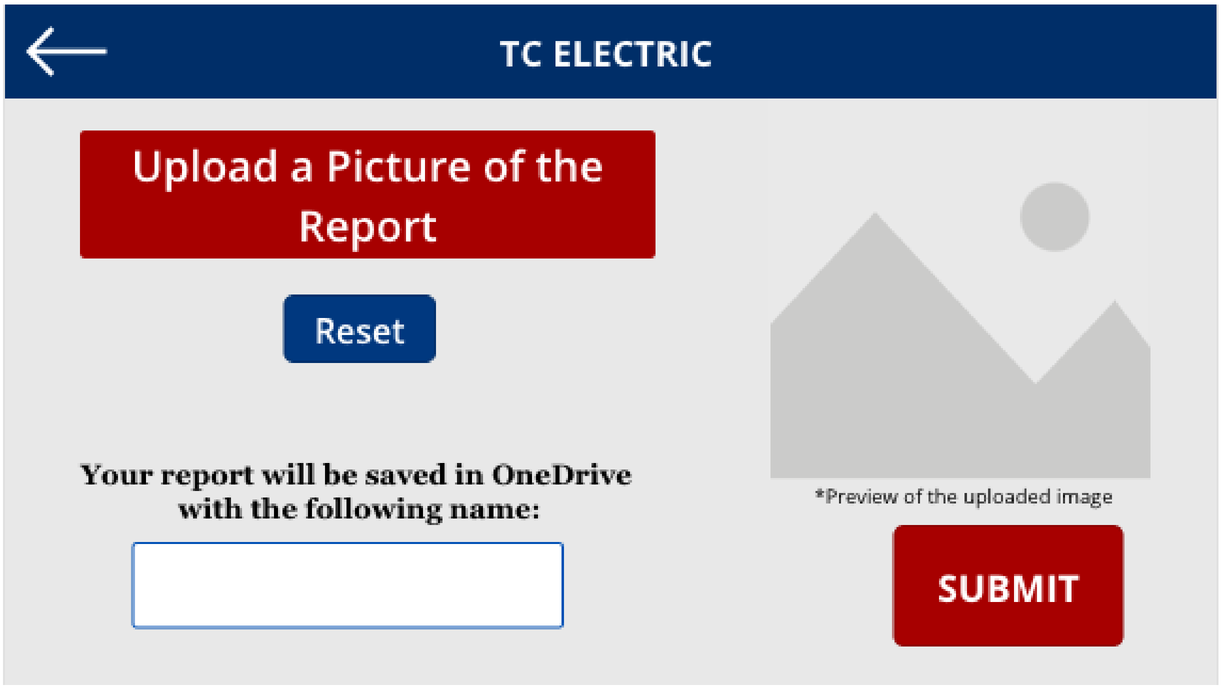

Inspection Report Records

No records found with the given selections,
go back and select 'Search with Just Job Name/Number'

Screen7: JobScreenEditExisting

This screen is very similar to the screen discussed before. This will show up if the user selected Edit an Existing Report from the welcome screen. Note that, in addition to all the flows that was used for creating a new report, this one has a hidden flow that is trigerred “OnVisible”. This flow is responsible for pulling any uploaded image in the original report, from OneDrive. Because this flow is trigerred OnVisible, it doesn’t require any user input.
Other than that, everything else is pretty similar to its create new report counterpart.

Screen8: UploadReportScreen


TC ELECTRIC
Upload a Picture of the Report
Reset
Your report will be saved in OneDrive with the following name:
*Preview of the uploaded image
SUBMIT

The main functionalities of this screen is very similar to uploading to pictures for questions discussed in CreateNewReport section. Let's look at the code for the submit button, all the actions happen inside of that button in this screen:

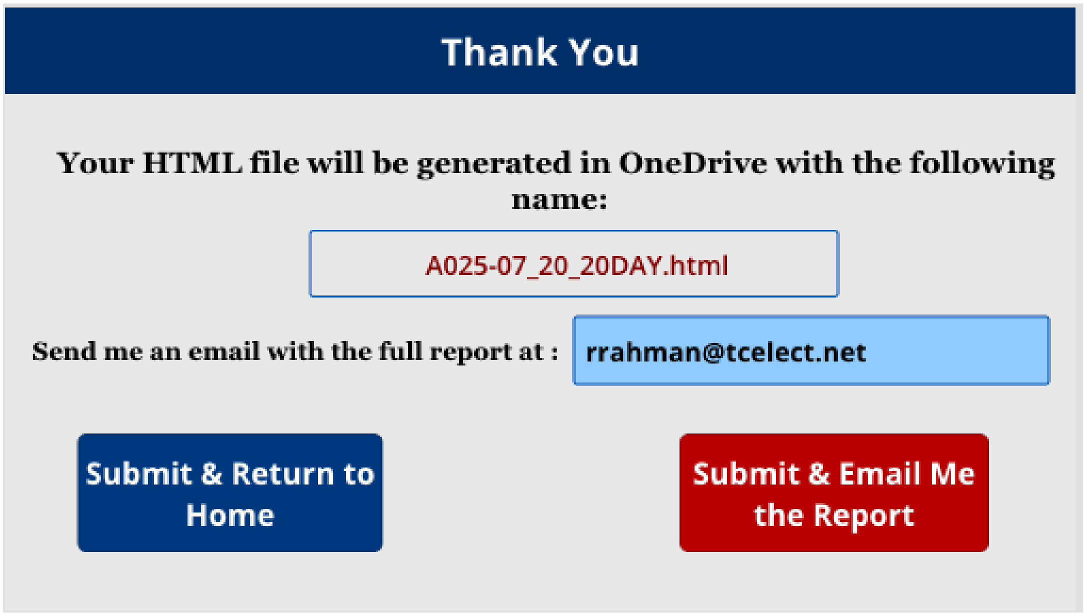
```
Set(BinaryImageData,
Substitute(JSON(uploaded_image.Image,JSONFormat.IncludeBinaryData),"""",""));
//Set(CompressedData,Substitute(BinaryImageData,Concatenate(First(Split(BinaryImageDat
a, ",")).Result,","),""));
ClearCollect(image_collection,{job:job,job_number:job_number,date:date,html_name:upload
ed_report_name,doc_name:TextInput1_2.Text,image_data:If(Text(uploaded_image.Image)="
appres://resources/SampleImage","No Image Uploaded",BinaryImageData)});
image_report.Run(JSON(image_collection));
Navigate(FinishScreen4UploadForm,Fade,{html_name:TextInput1_2.Text});

// THE FOLLOWING CODES ARE FOR DOCX ONLY //

//documentation: split the string until the first comma is found and substitute the whole thing
part with "")
// this will take care of all image type: jpeg,png, xml etc.
//(Concatenate(First(Split(BinaryImageData,",")).Result,","))
```

```
//Set(CompressedData,Substitute(BinaryImageData,(Concatenate(First(Split(BinaryImageDat
a,",")).Result,",")),"",1));


//ClearCollect(docx_collection,{job_number:job,date:date,docx_image:If(Text(uploaded_imag
e.Image)="appres://resources/SampleImage","No Image Uploaded",CompressedData)});


//image_report_docx_1.Run(doc_name,
JSON(docx_collection),If(Text(uploaded_image.Image)="appres://resources/SampleImage",
"No Image Uploaded",CompressedData));

//Navigate(FinishScreen4UploadForm,Fade,{html_name:TextInput1_3.Text,docx_name:Text
Input1_2.Text})
```

The top part of the code reads the base64 content of the uploaded image, puts it inside a JSON data structure and calls a backend flow called “image_report”. This flow is responsible for taking the image content and create a basic html structure with job, job name and ofcourse the uploaded image.

Then, you can see an ALL caps comment saying the following codes are for generating .DOCX type document. I will discuss more about it in the backend section of this documentation. But DOCX is something I experimented with and it requires a paid premium connector. I kept all my front and backend codes for this file type just so that we can reuse it if we ever decide to go back to DOCX type in the future.

FinishScreen: Send Email



Thank You

Your HTML file will be generated in OneDrive with the following name:

A025-07_20_20DAY.html

Send me an email with the full report at : rrahman@tcelect.net

This is the last screen that the user will see. It looks a little different for the Upload a Paper Copy route, nonetheless the functionalities are identical. The first box collects the name for the html file. It's a combination of the job number, job name, shift and the date. This makes sure we give a descriptive name to each file and they can be easily looked up while editing an existing report. Next, PoweApp collects the user's email address **automatically** (from the user's login info.). Finally, the user can choose to send himself an email with the html report attached or simply submit and go back to the home screen. The report will be saved in OneDrive anyway, so it's not imperative that the user has to get an email with the report. Let's take a look at the code for submit and email me the report button:

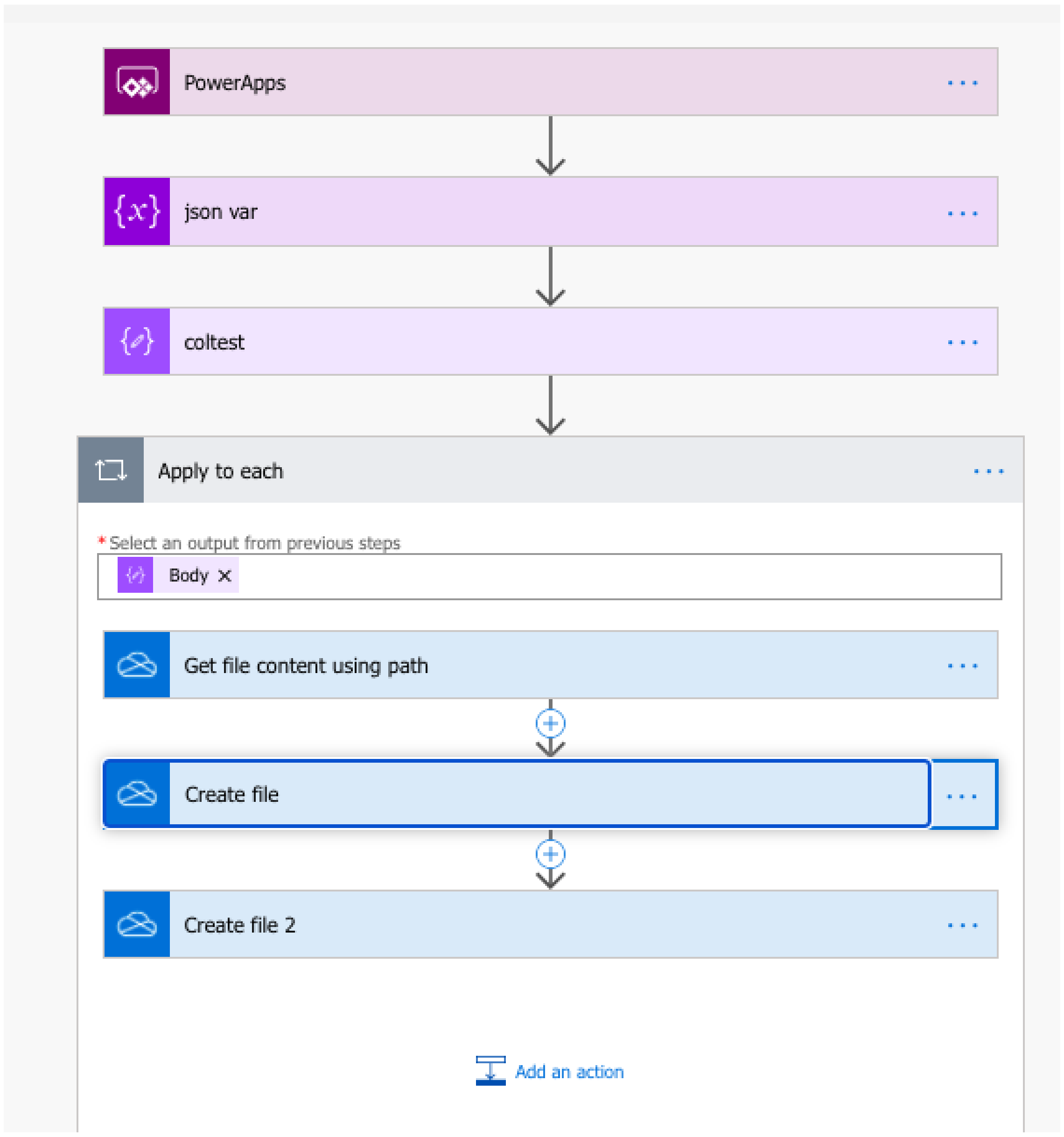
```
ClearCollect(email_dict,{filename:filename,email:email_address_1.Text});
email_flow_html_only.Run(JSON(email_dict));
SubmitForm(EditForm1_1);
Refresh(Table2);
NewForm(EditForm1_1);
Navigate(WelcomeScreen);
Navigate(WelcomeScreen);
```

The first line collects the filename and email address of the user and passes that onto the backend. Next,along with submitting and refreshing the form (from the previous screen), it triggers a flow in the backend for sending the email and then navigates to the welcome screen. Note the repetition of navigate to welcome screen is intentional. It sometimes leads to a buggy frozen screen if the command is put only once, no idea why it does that.

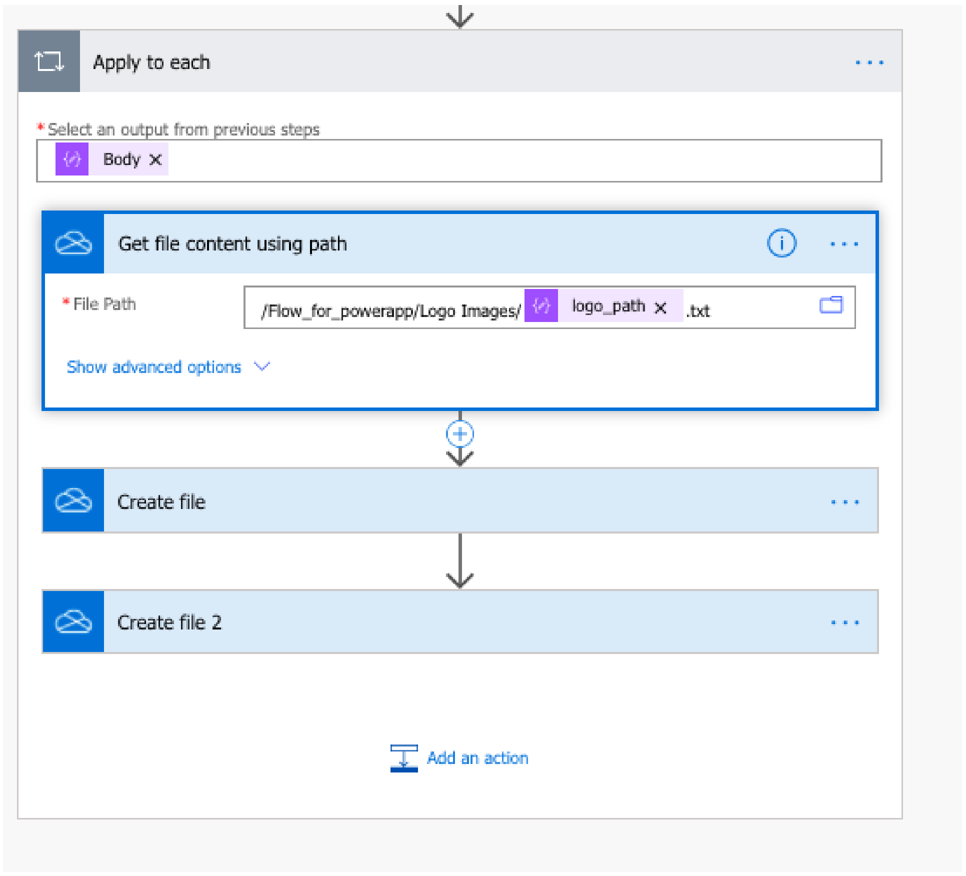
BACKEND : Microsoft Flow

FLOW 1: text_flow_html_only

This flow is one of the first and probably the most important one in the backend. This one is responsible for collecting all the user inputs from PowerApp user interface, convert the input as a json and then create an html file resembling the paper copy of the inspection report and finally populate the html skeleton with the user inputs. Let's take a look at the screenshot of the flow and then we will talk about some of the details.



In the purple blocks, we get the data from powerapp, create a json object to be able to refer back later. Finally in the Appy to Each, we collect the information for the logo that will be placed in the upper left corner of the HTML. The information "logo_path" comes from the Powerapp. We discussed before in the frontend section how the logos are numbered in 100,200,300 and 400. This information will now help us locate the right image from Onerive.

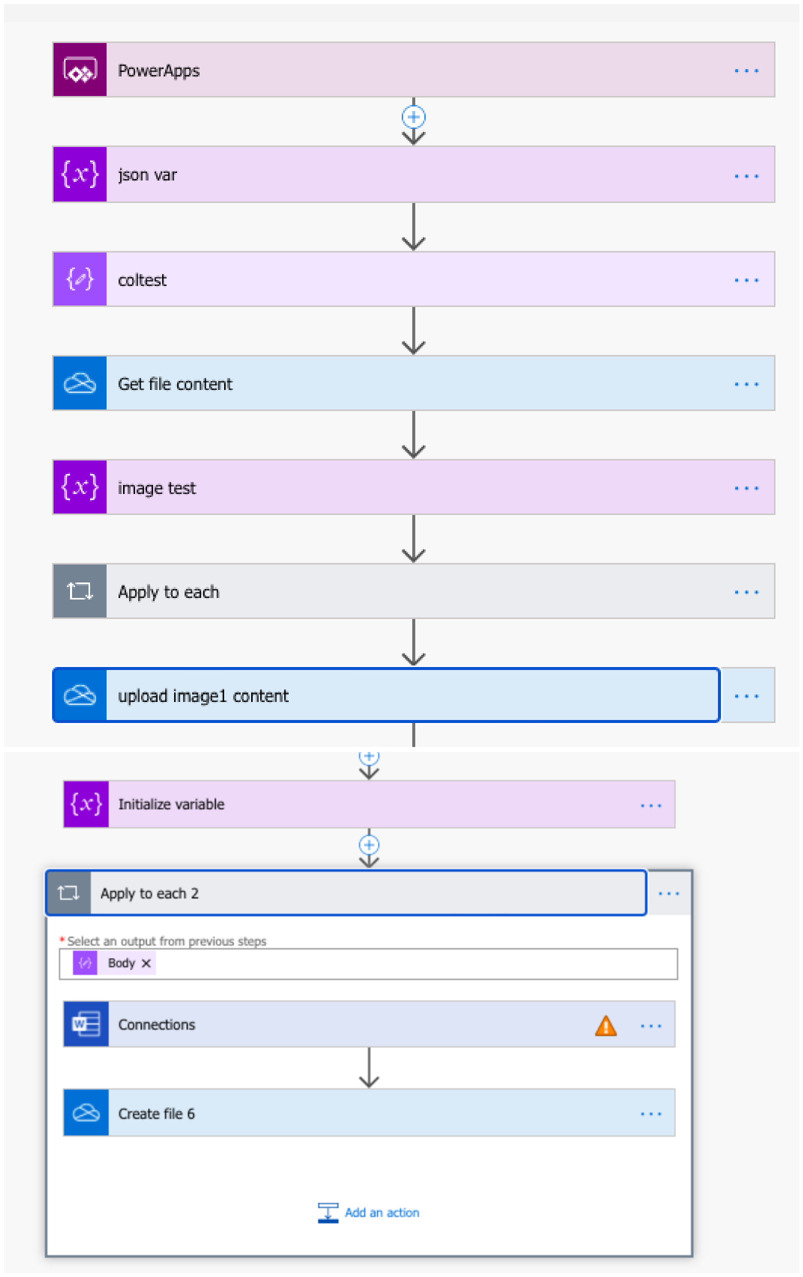

Apply to each
Select an output from previous steps
Body
Get file content using path
File Path
/Flow_for_powerapp/Logo Images/ logo_path .txt
Show advanced options
Create file
Create file 2
Add an action

In the create file section we put in the HTML code. In the create file 2 section, we create a .doc version of the inspection report (instead of .html). This solution was to avoid using html in the first place just so that the users can interact with a file type that they are more familiar with. However, as it turned out the images uploaded don't hold their original size and quality once the Word document is generated. So the create file 2 still generates a .doc file and stores it in the OneDrive, along with the html file. I kept it there just in case we come up with a solution to render better image quality on the doc version.

Flow for Word Type (.DOCX) File (with Premium Connector):

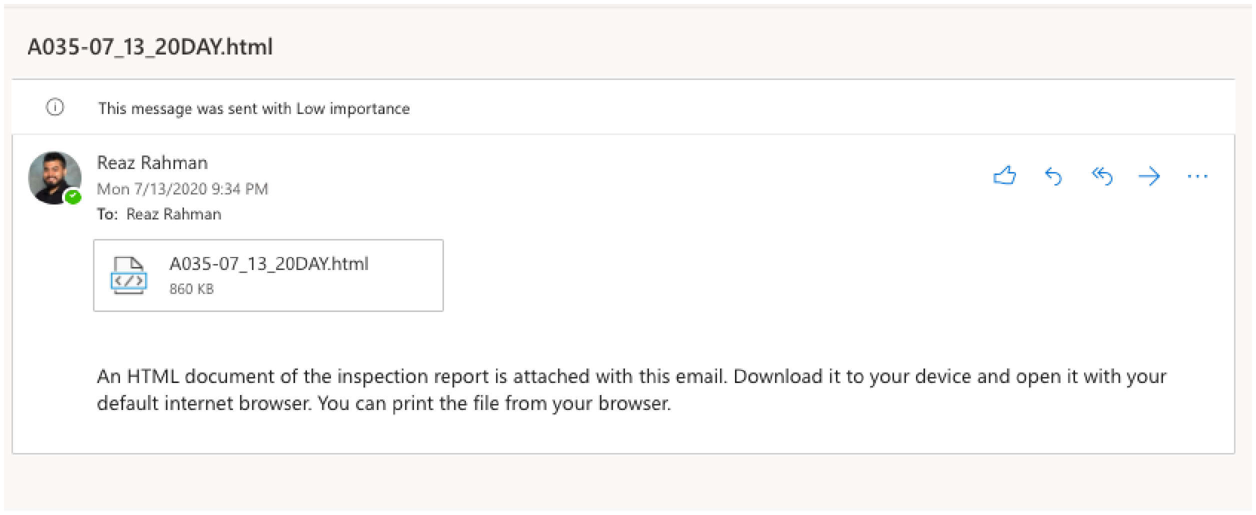
Speaking of that there is another flow in the backend called **text_flow** which also generates a word document of type .docx (instead of .doc). The same problem with the image quality that I mentioned for .doc type holds here as well. In addition to that, this one is dependent on a premium paid connector. I experimented with this flow and was able to successfully generate

Word files with uploaded images. So, I decided to keep it in the backend in case we want to use it later. Here is a compressed view of the flow:



FLOW 2: email_flow_html_only

This flow is responsible for sending the user an email with the html of the report attached with the email. Let’s take a look at what the email looks like:


A035-07_13_20DAY.html

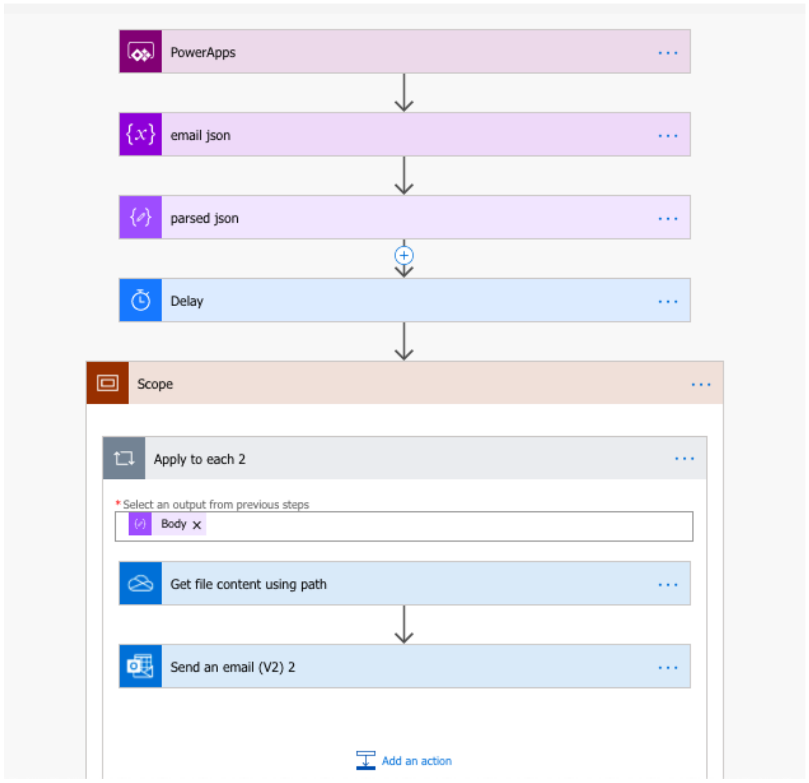
This message was sent with Low importance

Reaz Rahman
Mon 7/13/2020 9:34 PM
To: Reaz Rahman

A035-07_13_20DAY.html
860 KB

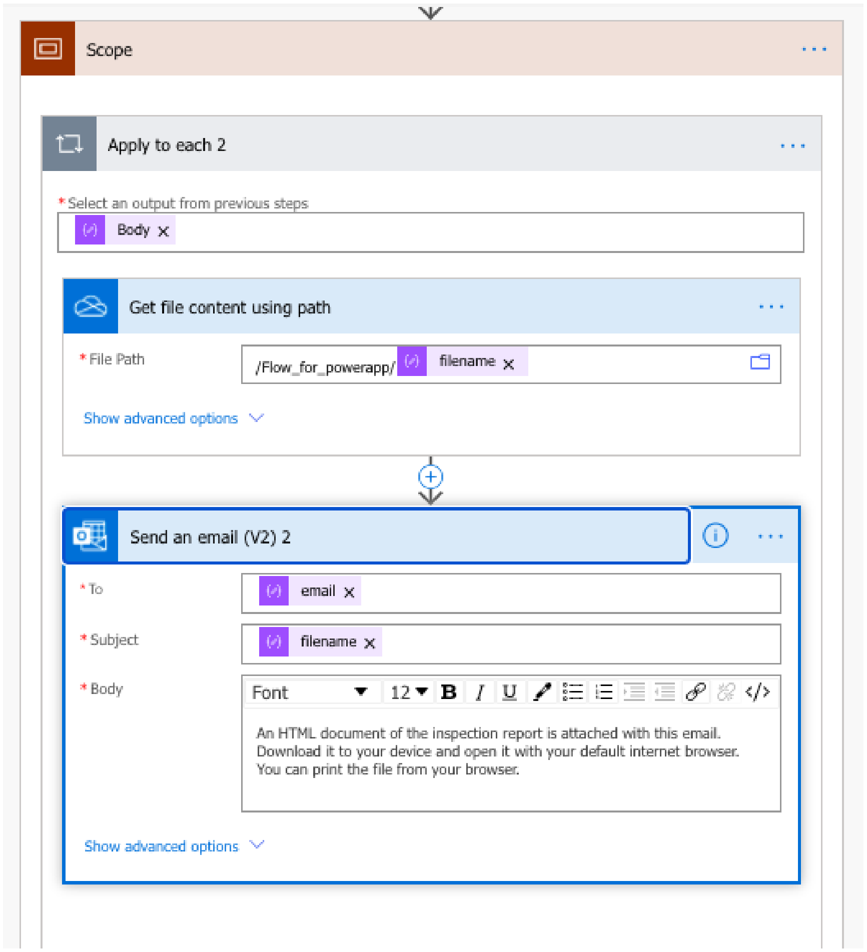
An HTML document of the inspection report is attached with this email. Download it to your device and open it with your default internet browser. You can print the file from your browser.

There is a simple instruction for the users in case they are not familiar with html file type. The subject line contains the name of the file so that it can be looked up easily. Now, let’s take a look at the flow:



Let's talk about some of the components here. The delay of a few seconds is put there just for good measure. Generation of the HTML and doc type document sometimes take a few seconds. And if the flow tries to access those files before they are generated, the whole flow will fail.

Now, let's take a closer look at the scope:

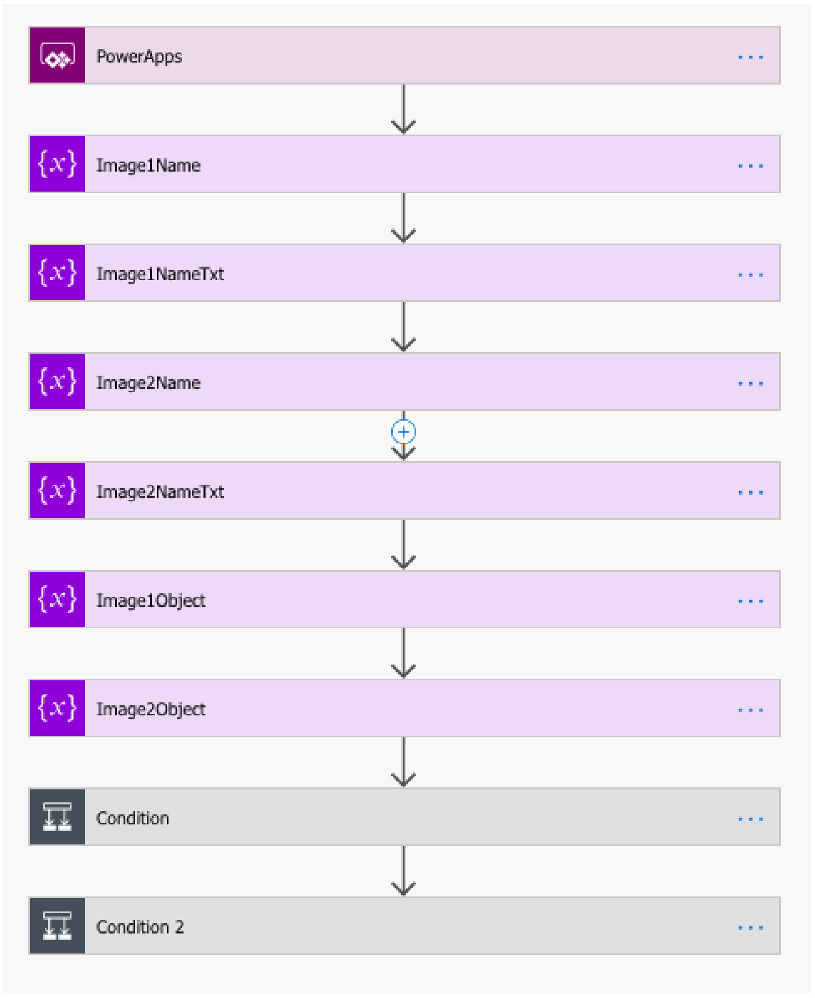


An HTML document of the inspection report is attached with this email.
Download it to your device and open it with your default internet browser.
You can print the file from your browser.

The inputs such as filename, email are collected from the powerapp user interface, more specifically in the FinishScreen of the app. The rest of the stuff is pretty self explanatory.

FLOW 3: New Report Store Image

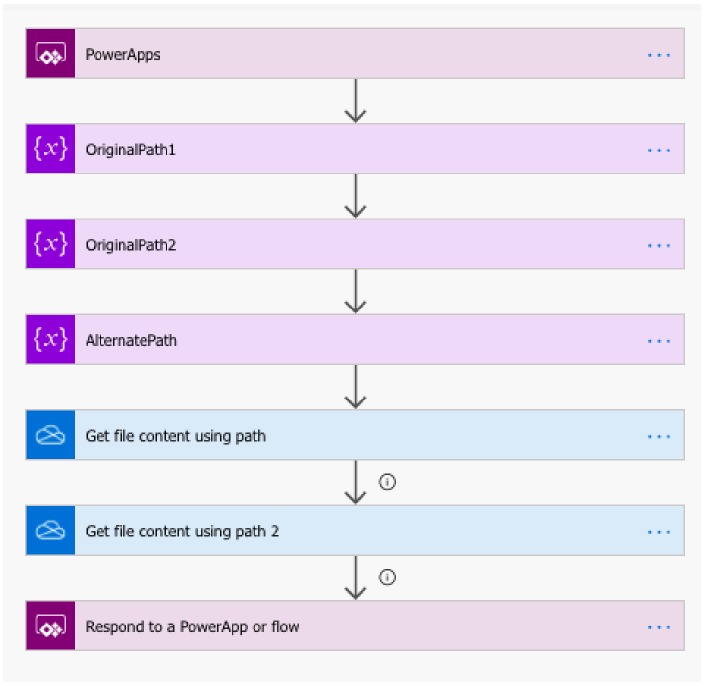
This backend flow is designed to store the user uploaded images from PowerApp into OneDrive. The images are systematically named so that while "Editing an Existing Report" the correct images can be grabbed from OneDrive and displayed to the user. This flow, however, is only for storing those images. In the next flow we will talk about how these images are read from backend and displayed to the user in the frontend.
Let's start by looking at the flow:



On a high level, it collects the names of the images and contents of the images from powerapp. Separate objects are created for each image. Remember the contents are in base64 format. Finally, the condition checks if the images are blank or not. If the user didn't upload any image for a particular question, then it will be a blank image. I put this part in the powerapp. The powerapp passes a "YES / NO" variable in the backend. Yes suggests an image is uploaded and NO suggests no image was uploaded.

FLOW 4: ShowImageFromOneDrive

This flow is responsible for collecting the image content from the backend and display it to the user when they select Edit an Existing Report. In order to do that properly, it needs the full image path (in OneDrive). That information is stored in the excel file when the user created a new report. When this flow is trigerred the full path is collected from the excel file and used to render the images from backend.


PowerApps
OriginalPath1
OriginalPath2
AlternatePath
Get file content using path
Get file content using path 2
Respond to a PowerApp or flow
This is what the flow looks like. OriginalPath1 and 2 are the full path for finding those two images in OneDrive. The alternatepath is no longer used in the most updated version of the app. Then the image contents are obtained by using the path and in the final step the two images are fed back into power app.

<u>How is this flow trigerred?</u>

This flow is hidden under "OnVisible" property of the **Edit An Existing Report Screen**. This makes sure that everytime that screen loads on powerapp, this flow is automatically called without having to rely on any user input.