**Technical Design documentation/Architecture**

Text Extraction from WCF documents:

Architecture:

Diagram

Description automatically generated

Convert pdf to images:

Library used: py2pdf

A picture containing text

Description automatically generatedDependency: poppler

The above code converts pdf into an image and saves the images in order of the pages.

Extracting text out of the Pdfs:

Methodologies used:

**Without converting the pdfs to images:**

1.pdfplumber

2.pdfminer

3.pymupdf

Problems with these libraries:

* Since the Claim forms are scanned documents, these libraries weren’t able to extract the data from the forms.
* To solve the above problem we first converted the forms to searchable pdfs.
* With this we were able to get the data out of the pdfs, but it was not very accurate.

Text

Description automatically generated

The above code snippet converts a scanned pdf document to a searchable pdf.

The results were similar for all the libraries hence we did not continue with these libraries.

**After converting the pdfs to images and using OCR engines**

1. Pytesseract:

Library : pytesseract

Dependency : tesseract OCR setup (path is required while running the code)

With this library results were satisfactory, but the extracted text also contained some garbage text that needed to be managed.

Graphical user interface, text, application

Description automatically generated

The above code snippet extracts the text from an image and prints the text , text\_fin\_1 contains the extracted text.

Since tesseract had multiple segmentation modes to print the text so tried all those.

The attached link contains information about the seg modes.

<https://pyimagesearch.com/2021/11/15/tesseract-page-segmentation-modes-psms-explained-how-to-improve-your-ocr-accuracy/>

In the above code snippet, psm3 (seg mode) is used.

In tesseract psm3 and psm6 were giving the best results.

Since the results contained garbage data and accuracy was not very good, so we didn’t continue with this approach.

Easyocr (OCR engine)

Library: easyocr

Dependencies:

While running easyocr the kernel becomes dead sometimes, to handle that error

*os.environ['KMP\_DUPLICATE\_LIB\_OK']='True'*

following code snippet needs to be added in the code before importing easyocr.

This OCR engine gave the best results in comparison to above engines and libraries but the runtime was a bit slower.

Since easyocr had the highest accuracy, hence we finalized this approach for text extraction.

**Text extraction final approach**

Step 1: Processing the image

Firstly, we take the image and apply thresholding and grayscale on the image using opencv.

The code snippet attached below performs the step 1.

Text

Description automatically generated

Step 2: Detecting horizontal and vertical lines

Then we detect horizontal and vertical lines in the forms using kernels, the size of the lines detected can be controlled using kernel sizes. If the kernel size is small it will even detect small lines, so in that the horizontal lines in the alphabet can also be detected, so we are currently using higher kernel sizes.

Text

Description automatically generated

The code snippet attached is taking care of this part where kernel\_v and kernel\_h are horizontal and vertical kernels respectively. [Click here](https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html) for morph documentation.

Step 3. Combining the horizontal and vertical lines.

Text

Description automatically generated

The above code snippet is combining the detected horizontal and vertical lines, with the code in the first line we can control the density of the final lines so if we’ll increase the values in the bracket (5,5) the density of the detected lines will increase.

Step 4. Saving the coordinates and image boxes

In this step first we are storing the x and y coordinates, length and width of each detected box and then we are using those coordinates to crop out those boxes and saving those image boxes in img[] array.

Text, application

Description automatically generated with medium confidence

The above code snippet is performing this step.

Step 5. Extracting the text (required fields)

In the last and ultimate step, firstly we are getting the indexes of the required image boxes by trial and error and then applying easyocr to extract the text out of those boxes and then storing that information in a dictionary.

Text

Description automatically generated with medium confidence

Here the numbers passed in the functions are the indexes of the image box in which that data is present.

Then we are storing all the information in a dictionary using the patient’s name as the key.

**Automation of WCF portal**

Architecture:

Timeline

Description automatically generated

The above attached picture represents the solution architecture of the WCF automation that we did. Firstly our program takes input from the excel sheet , then logins in the WCF filenet portal , then searches for the attachment using a search logic (attached below) then downloads the attachment if it is present , then it extracts the data from the downloaded document and matches 4 fields Patient name, Patient dob, service date and Provider’s name and returns true or false based on the matching.

**Methodologies/ Approaches used :**

We used selenium and started working on automation. We searched how to use it for automation and tried to open the WCF Portal in Chrome and it was working fine and FAST and reached till the DCN / HCID searching place and then,

Problem- couldn’t get the information for patient on chrome

Solution- came to know that WCF website is Internet Explorer based website, and we need to open the WCF website on IE in order to access the patient's uploaded documents.

So, we shifted to IE and we ran the same code on IE,

Problem- It wasn’t working and we looked for reasons why it wasn’t

After looking into google long enough, we suspected that WCF opens in second window when we open it while it opens in one window in chrome, so maybe the code is executing for 1st window on IE which doesn’t have the portal (login credentials are needed on 2nd window). We tried to test this guess by looking for code to switch window and then automating login using that chrome code which we had already written.

After getting the code for changing window, we were successful and our suspicious was right and we could login. But after login, we couldn’t select the ‘claims corro-extended search’. Href we got using inspect element was correct only and still we didn’t know what was not working. We looked on the internet but didn’t get the solution and were clueless. Later, we had a meeting with Hari and there he told us to use TAB and ARROW KEYS instead of inspect element to navigate to the desired fields of ‘claims corro extended search’ in both ‘Role’ and ‘Search’. Then, we calculated the number of TABs needed to navigate to ‘Role’ and ‘Search’ for the desired field for searching. Then, also we were facing issues and those arrow keys weren’t working after doing tabs, there we tried this code on some dummy websites and saw that code was working fine. We speculated that after selecting ‘claims corro-extended search’ in role, there’s a refresh of IE browser which gives a lag in time and we started searching if we can account for that lag in our code. From there, we found about sleep time and incorporated it in our code which made it work. We were finally able to search for each patient’s documents using DCN number and HCID.

But when coming back to search using HCID after finding nothing using DCN number, we click reset button and then enter HCID. The reset button was not being clicked using selenium driver and we cross checked it again and again. While running the code again and again, we can see that driver was also not selecting the browser button while searching for documents as well. It was selecting - for both browse and rest button action but for browse, we got the desired action of opening the document collection area so, it is fine but for rest, it was again opening the doc collection area instead of resetting, so we must do something about it. We took a call with Hari on Friday evening and he looked into the problem and asked us if we skip this reset button pressing and simply do a backspace to erase the DCN number entry and write HCID then on. We could do that successfully and finally we could start on automating the process of viewing and downloading documents but we were still doing it for a single person because more roadblocks may come if we start it for all excel data. We tried opening the documents in our systems on WCF portal but couldn’t open it even manually trying it.

We tried automating doing multi-threading but that did not work as this whole automation is a synchronous process and not asynchronous. ‘Downloading’ thread and ‘document checking’ thread was not working in proper order because of sleep time used in download thread. Threads were being ‘document checking’ thread was being run simultaneously with ‘download’ thread because of encountering sleep time in each thread, so every time we run the code, it gives ‘Medical records not found’ output because it was checking document and not waiting for the document to be download, so it doesn’t find the required document in the directory. It is like a relay race where every runner runs as a team by passing batons to the front runner of his team. We can say that we can't run two runners simultaneously like multi-threading because it's not a sprint race. We need every runner to wait for the previous runner to finish its distance and pass the baton and then, he/she will start running the next defined distance to do the same for other runner waiting for him/her. Our code also works sequentially, and every step must wait for the previous step to finish completely, any error at any step will spoil everything. We need every individual step to speed up in order to reduce the overall time consumed to check for medical record.

Diagram

Description automatically generated

EasyOCR also takes lot of time to check for medical records, but this library works fast for GPU, and we can run the code on GPU instead of currently running on CPU. This will reduce a lot of time which gets consumed in checking the medical records for each downloaded document from the WCF portal.

We were facing issues with downloading as this process was totally dependent on the amout of sleep time we gave to WCF portal. We were using 30sec sleep time as upper bound for downloading because downloading the pdf from wcf was very slow and we must give an upper limit for sleep time so that next step of checking the pdf for medical record can be done. If the pdf takes longer time and breaches the upper limit of sleep time, then downloading fails and check the document also fails, ultimately returning in 'Medical Record not found' as output. We also came to know that these attached pdfs might go on to 100 pages and those pdfs will certainly need more sleep time, maybe 5min and we need to sort out this problem, we cannot allow very large sleep time just for few exceptional pdfs, this will reduce the response time of our code and waste lots of resources, money and time. We asked Bharti if it's ok if we download only a certain number of pages only and not whole big pdf and he said that only starting pages will suffice to check for the required fields and make a deduction of it being medical record or not. This helped us to decide an upper bound for sleep time needed for downloading but the challenge was to find we can find total number of pages in PDFs before downloading. After meeting with Sriram and doing a lot of brainstorming, we got a new way where we can only download starting few pages instead of whole pdf.

To optimize the whole process, we finally used multithreading for the text extraction and medical record validation part.

Hence the overall workflow will be

first check and download the valid records using DCN and HCID then checking whether the attachment is a medical record or not using multithreading.

**Code overview:**

Step1. Importing all the required libraries

Text

Description automatically generated

The above code snippet imports all the required libraries:

Selenium – to interact with the site and perform actions

Pyautogui-to interact with the pc when driver loses control in the website

OS – to get the working directories

Threading – to perform multithreading

Datetime – to get current date time which we are using in the naming convention of the files.

Step 2. Taking entries from the [excel sheet](https://mysite.wellpoint.com/personal/ai01475_ad_wellpoint_com/Documents/Desktop/scraping%20project/WGS%20DATA.xlsx?web=1):

Firstly, we are taking entries from the excel sheet as input to our script, the fields taken are DCN, HCID, Patient Name, Date of birth, service date, provider’s name NPI and creation date.

Text

Description automatically generated

The above attached code snippet is taking care of step 2.

Step 3. Logging in the portal:

The second step is to login in the portal, when we open the filenet portal using selenium 2 windows open, one is it’s start page and the login page opens in another window , since the driver has the control of the current window, we switch the control of the driver to another window by getting the id’s of the both the windows and then switching to another window by driver.switch() method and then we login in the portal.

Text

Description automatically generated

Step4.Selecting the dropdowns and searching through DCN

Then the code Selects claims\_corro\_extended\_search option in both the dropdowns and searches through DCN.

Text

Description automatically generated

Step5. Searching for the medical record:

Below attached is a flowchart explaining all the necessary steps performed by the script while searching for the proper attachment.

Diagram

Description automatically generated

Lucid Chart

Step6. Extracting text and Validating Medical Record

The naming convention followed to save the downloaded medical record is

DCN,20DCN\_current\_date\_time

We followed this naming convention so that we don’t have to rename it again and again.

After downloading and saving our program first gets the list of all the documents having name starting with a particular DCN, then using multithreading it forms multiple threads and then checks for each thread simultaneously whether it’s a medical record or not.

Graphical user interface, text

Description automatically generated

The above function returns the list of the documents whose name starts with the DCN number passed in the function.

The code snippet attached below takes care of the process of firstly identification of the medical record and then matching the 4 fields

Medrecord() function takes 4 parameters Patient name, Service Date, DOB and Provider’s name as input.

Firstly it checks if the form is a claim form by extracting the data out of specific coordinates, if it is a claim form it returns false and if not it performs the steps mentioned below:

Firstly it extracts the data from the first 5 pages one by one and simultaneously matches the fields, firstly it converts the extracted list from the page to a string then it removes all the spaces from the string and then converts the whole string to lowercase letter, and also the Patient’s name and provider’s name to lowercase letters and then it searches for these fields in the final string.

For the birth date and Service date it searches for them in the final string in two formats

DD/MM/YY and DD/MM/YYYY if date is matched in either of those formats it increases the counter similarly for the Patient and Provider name field.

If the counter value at the end is 3 , it returns true else it returns false.

Text

Description automatically generated