Documentation

Step 1: Getting JSON Files

I proceeded with installing and using Adobe PDF Extract API. Using its **private.key**, **pdfservices-api-credentials.json,** and **extract_txt_table_info_with_table_structure_from_pdf.py** files I got a ZIP file containing JSON format of sample pdfs given, which will be further used to extract data from it to give the final ExtractData.csv file.

Problems: I used this **extract_txt_table_info_with_table_structure_from_pdf.py** file so that I can get a direct table structure, but it is only optimal when JSON format is well written which is not likely in our case. **To make CSV file right We have to write our own code. I am going to use Python.**

Sources: DC Integration Creation App (adobe.com) -> To get credentials and .py files to use API.

Step 2: Finding few common things between JSON Files, so that a single code will run correctly on all of them

JSON files were not properly and orderly generated by API

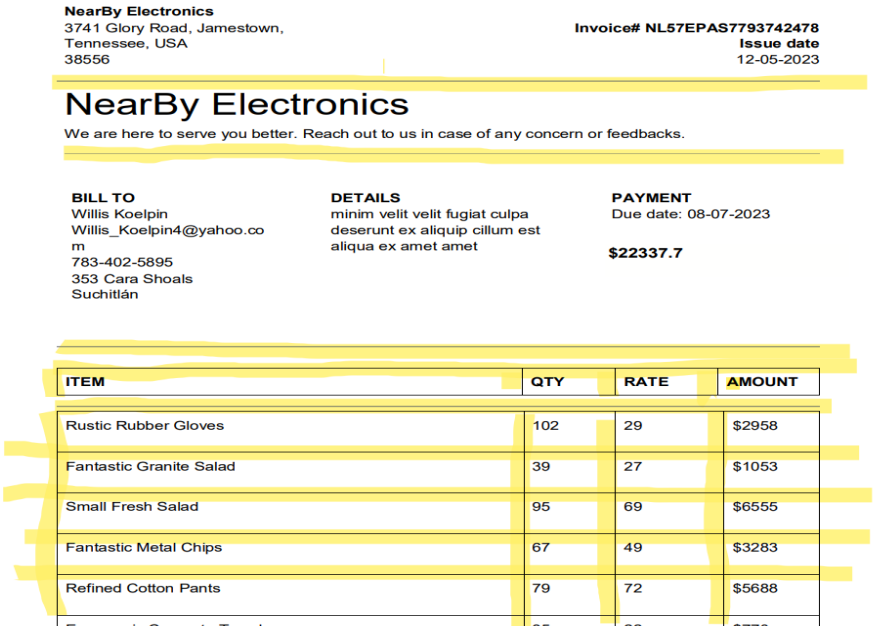
Finding out that the JSON files are not in proper sequencing, I expected that just reading line by line JSON files text and adding to CSV file accordingly will solve the problem.
But those JSON files were not in proper ordering of sequencing, they are quite randomized.
So after knowing this only option left is to write some code to sequencing the randomization as well as putting the same information in CSV file.

Even when JSON files were randomized still there are some parts where they are well ordered, Analyzing those well ordered parts and and making the unordered parts well ordered using code will make overall structure well ordered which can be processed into CSV file.

After analyzing the way API is actually working on well ordered parts I found out only these 3 levels of solving problem:

1. **API IS ABLE TO DIFFERENTIATE TEXT BETWEEN LINES AND TABLES.**
   These horizontal lines in PDFs help me to differentiate between texts to some extent, as I can make sure some text generated by API in JSON is going to start from “BILL TO ” and not some random other text. Similarly for “ITEM ” as well.

   But still I have to differentiate between text sandwiched between THESE LINES.



NearBy Electronics
3741 Glory Road, Jamestown,
Tennessee, USA
38556

Invoice# NL57EPAS7793742478
Issue date
12-05-2023

NearBy Electronics

We are here to serve you better. Reach out to us in case of any concern or feedbacks.

BILL TO
Willis Koelpin
Willis_Koelpin4@yahoo.com
783-402-5895
353 Cara Shoals
Suchitlán

DETAILS
minim velit velit fugiat culpa deserunt ex aliquip cillum est aliqua ex amet amet

PAYMENT
Due date: 08-07-2023

$22337.7

ITEM	QTY	RATE	AMOUNT
Rustic Rubber Gloves	102	29	$2958
Fantastic Granite Salad	39	27	$1053
Small Fresh Salad	95	69	$6555
Fantastic Metal Chips	67	49	$3283
Refined Cotton Pants	79	72	$5688

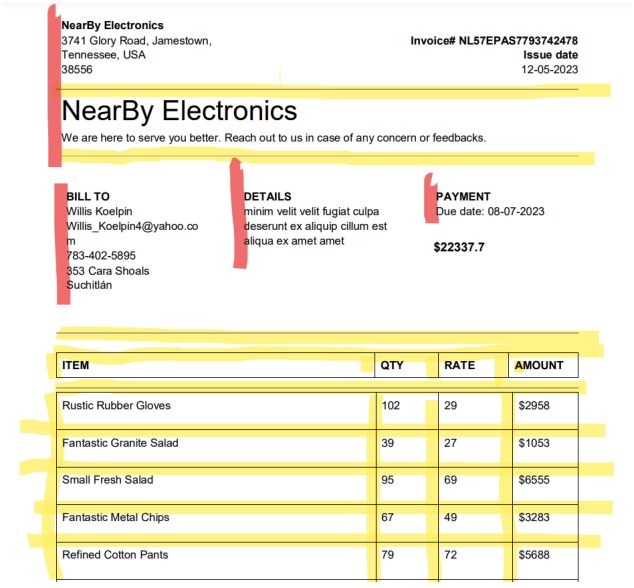
2. **TO DIFFERNTIATE TEXT BETWEEN THESE LINES. Use of Bound variable for same text, as it will remain fixed as long as format of Receipt is changed**

   Thinking of many variables so that I can find something common in all json files such that

   To differentiate text between lines we can take advantage of **Bounds variable in JSON file,** which actually show the position of text under some imaginary rectangle. As Format of Receipt is going to be same. For example BILL TO will remain there at that position, even there are different receipts.

   So, details below them also started vertically below them, hence there left position are going to remain same. Similarly for all text between lines. Hence we will be able to differentiate text between lines as well.

   As shown in figure using red lines these are some in between text that are going to be almost properly vertically aligned as its topic variable. Referring topic variable such as BILL TO, DETAILS, PAYMENT etc. If we found out mathematical position of these topic texts we will be able to differentiate between lines as well, and that is the same provided by Bounds variable in JSON file.



NearBy Electronics
3741 Glory Road, Jamestown,
Tennessee, USA
38556

Invoice# NL57EPAS7793742478
Issue date
12-05-2023

NearBy Electronics

We are here to serve you better. Reach out to us in case of any concern or feedbacks.

BILL TO
Willis Koelpin
Willis_Koelpin4@yahoo.com
783-402-5895
353 Cara Shoals
Suchitlán

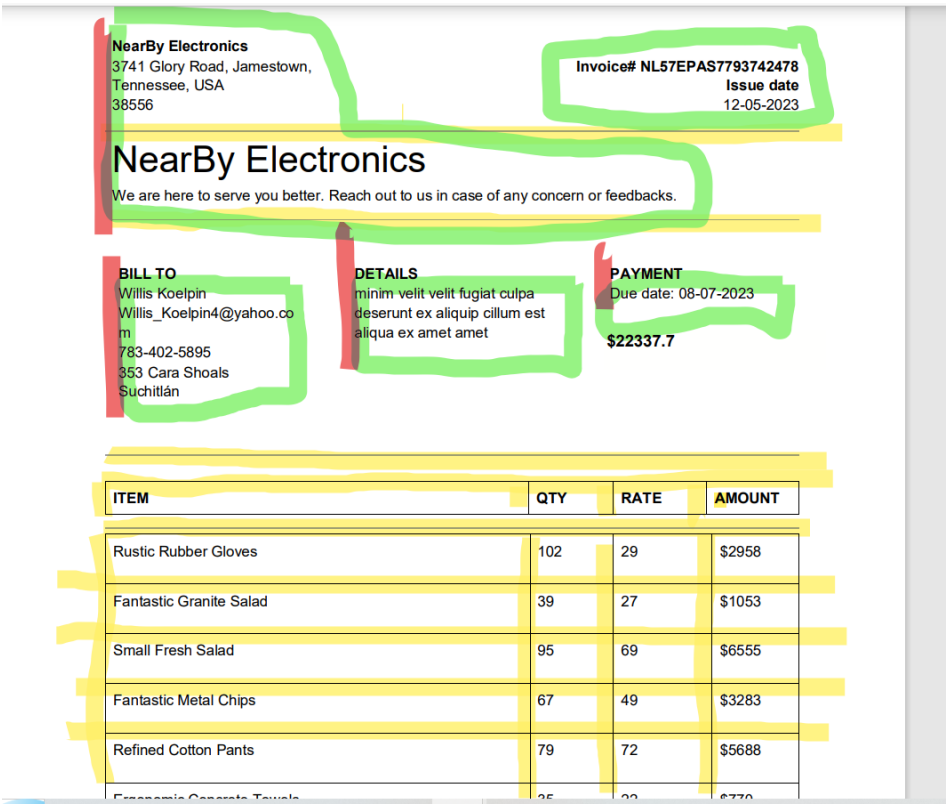
DETAILS
minim velit velit fugiat culpa deserunt ex aliquip cillum est aliqua ex amet amet

PAYMENT
Due date: 08-07-2023

$22337.7

ITEM	QTY	RATE	AMOUNT
Rustic Rubber Gloves	102	29	$2958
Fantastic Granite Salad	39	27	$1053
Small Fresh Salad	95	69	$6555
Fantastic Metal Chips	67	49	$3283
Refined Cotton Pants	79	72	$5688

3. **Now using the format of Receipt we can find our main details:**
   For example: in BILL TO Title: First came the NAME, then EMAIL, PHONE NO., STREET ADDRESS and at last CITY
   Using text we get after 2nd point, we can get these as from 2nd they will be under a single string of text under various Topics Variable.



NearBy Electronics
3741 Glory Road, Jamestown,
Tennessee, USA
38556

Invoice# NL57EPAS7793742478
Issue date
12-05-2023

NearBy Electronics

We are here to serve you better. Reach out to us in case of any concern or feedbacks.

BILL TO
Willis Koelpin
Willis_Koelpin4@yahoo.com
783-402-5895
353 Cara Shoals
Suchitlán

DETAILS
minim velit velit fugiat culpa deserunt ex aliquip cillum est aliqua ex amet amet

PAYMENT
Due date: 08-07-2023

$22337.7

ITEM	QTY	RATE	AMOUNT
Rustic Rubber Gloves	102	29	$2958
Fantastic Granite Salad	39	27	$1053
Small Fresh Salad	95	69	$6555
Fantastic Metal Chips	67	49	$3283
Refined Cotton Pants	79	72	$5688

Step 3: Writing Python Code

Using step 2 conditions we will be able to write a code which will generate CSV file with Extracted Data.

EFFICIENCY OF MY CODE

1. Using JSON library and extracting whole JSON in single variable is going to cost us Runtime Space of JSON file**(generally 50 KB).**
   But I used IJSON library which will extract single line of IJSON at one time and as we will be only storing text part in our code it will take only Runtime Space of text written (**generally less than 1 KB**). As analyzing the text, it is going to have normally around 1000 characters or less than that.

2. As one receipt is getting read only once hence **time taken by code is going to be one of the best**.

3. Many of the problems in API for example giving JSON format in uneven way is solved using Bound variable, which will cover **many edge cases as well**.

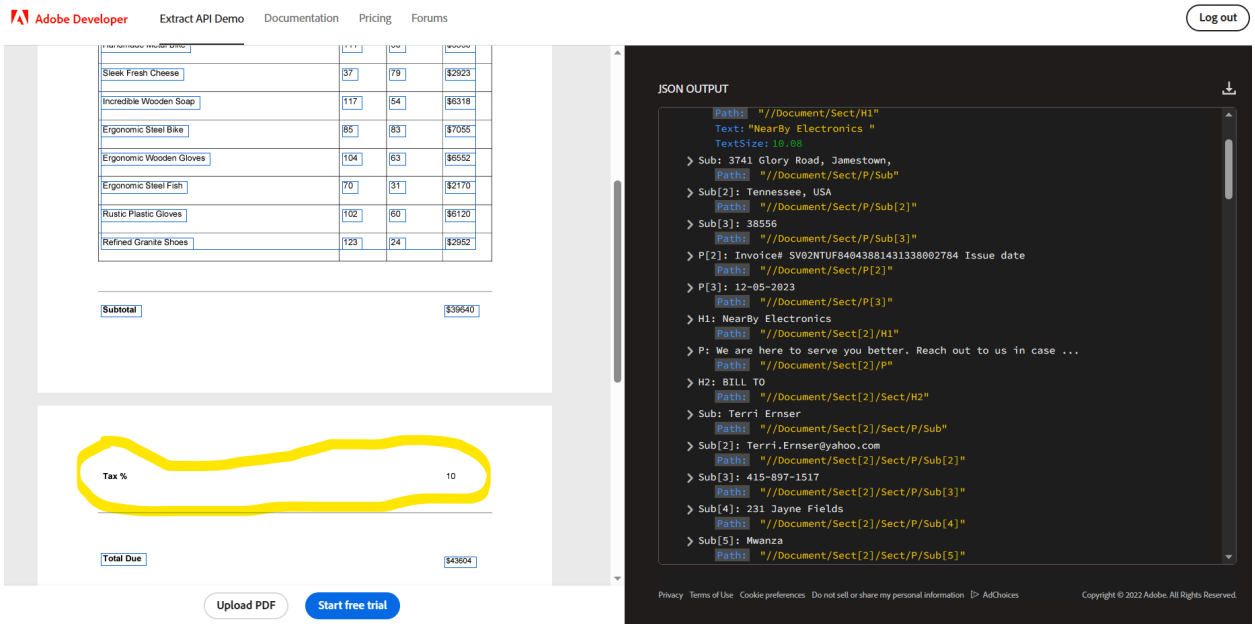
4. **Code is Universal if particular Format of receipt is fixed**. Even when Other than Nearby Electronics some other company would be there then it will also work fine.

WHAT COULD BE BETTER IN CODE

1. Although code is going to cover almost all cases, But Step 2, 3rd part can be modified to make it more robust. As email, date, Street Address has some particular format that can be detected using **Regex.** Which will perfectly detect these details and will have almost zero chance of wrong output.
   Also the City can be found using **GEONAMES API**, which will help in finding City in long text string.

A Sample PDF where API fails to Detect

(output81.pdf)



To remove this discrepancy, I added if it fails to find tax value then put 10.

THANK YOU FOR ADDRESSING MY WORK

TANUJ KAMBOJ
BTECH.
2021-2025
IIIT-D