MINT 709

Capstone Project Report

Based on

Email to REST (E2R) system

Instructor: Prof. Paul Lu
Presented by :Sachin Kaushik

Dated:11 th Apr 2018

--:INDEX:--

Page No.

Acknowledgement

I would like to take this time to thank Prof Paul Lu for special guidance, patience and support provided throughout to complete this project. It has been such a privilege working with him. This project would not have been possible without his help and insight.

The journey in doing this project and completing my degree has been a great experience and one I take pride in accomplishing.

I would also like to thank my classmate Wenting Zhang for her project proposal which helped me at the time of initial stage of my project.

Abstract

A variety of online systems such as e-commerce sites Shopify, mailing list like Google News Alerts are designed to interface to the world by sending an email.

In this project, I worked on Email-to-REST (E2R) model. The basic idea of Email-to-REST (E2R) is to form a programming model and associated implementation of its components for processing and generating online orders through an email from clients. Thereby, it can be called a Client and Server model. Client-Server model is a distributed application structure that partitions task or workloads between the providers of a resource or service called server and service requesters called clients. A Client does not share any of its resources, but requests a Server's content or service function.

The email from Client should be well-formed. If email is non well-formed, there would be problem in finalize the request. However, during specific conversation email could also be non well-formed e.g. asking for further unique information to finalize the invocation. In last, the request email will result in an invocation or call to a server using a representational state transfer (REST) interface. Therefore, the E2R system is a REST-based direct message-passing system.

This project focuses on the well-formed components of the E2R system which involve mainly Mail Praser, Q&A (Questions and Answer) , State Storage and REST Invocation.

1. Introduction

A variety of online systems such as e-commerce sites Shopify, mailing list like Google News Alerts are designed to interface to the world by sending an email. Great motivation behind the designing such a system is to fully automate a response using the email parsing and entering the ordered data into a storage system via REST Invoker.

The implemented E2R system automates processing of order emails from Shopify and enters it into the computerized order system. More generally, sending an email to invoke an action on a server is a useful interface since it is sometimes easier to send an email from a variety of different devices (e.g., smartphone, desktop, laptop, Chromebook, public access machine (when appropriate)) than to invoke a script or run a program to invoke the REST interface directly.

Representational state transfer (REST) or RESTful web services is a way of providing interoperability between computer systems on the Internet. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations.

The basic idea of an E2R system is similar to mailparser.io and docparser.com, which automatically parses emails with pre-designed algorithms, a semi-structured layout and then invokes an appropriate action (e.g., extracting user-specified data) on a back-end server.

The goal of this project is to implement a fully functional E2R system having main focus on Mail Q&A and State Storage system other than Mail Parser and REST Invoker (Figure 1: Architecture of E2R system). Mail Q&A is a special part of E2R system which provides quality and automatic features to the web service of e-commerce companies.

We take Shopify as an e-commerce site for example. It is important to note that Shopify is a virtual platform for many different types of shops such as for Garments, fashion, sports, books etc. Let us take a scenario. Let us assume, one client say, I want to order a shirt using Shopify. I went to Shopify website, selected the designer shirt say, from Eureka shop (fictional shop), entered my desired selection for shirt and tried to place an order (Figure 1).

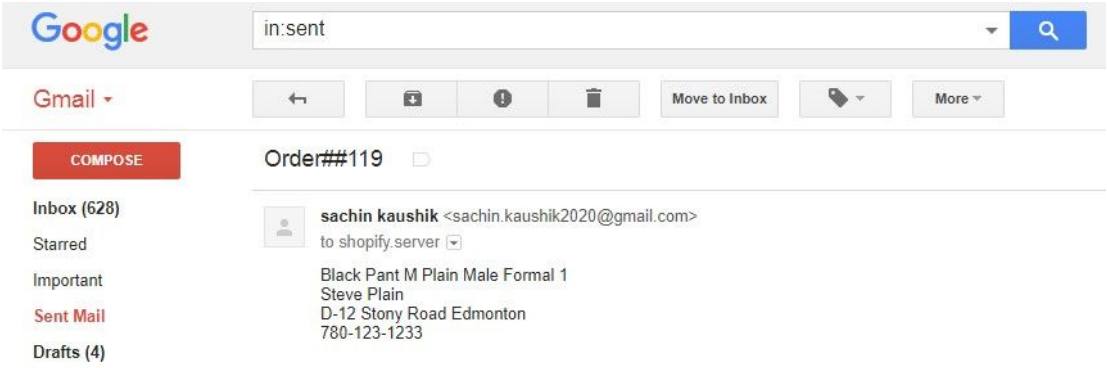


Figure 1: Email request (Client)

In that process, Shopify platform passed to Mail Watchdog (Figure 4). Mail Watchdog will monitor the email box and respond when a new email is received. The Mail Watchdog reads the email contents of email sent by Shopify.com that are in human language which has information of item requested by me. Mail Parsing operates through a specific algorithm. It takes all important information from the email received from Shopify email such as size, color, type, quantity, design etc. and passes control to Mail Q&A (Figure 4). The useful information of this email is parsed by Mail Parser (Figure 4) .

Whenever there is any missing information to finalize REST Invocation, Mail Q&A generates that specific query for clients through e-commerce website. After this, client can answer that query which will again pass to E2R system to complete REST Invocation and to confirm the order. For instance, if I entered parameters like type for shirt does not found or matched in the system then Mail Q&A will generate query for client asking about type for shirt (Figure 2).

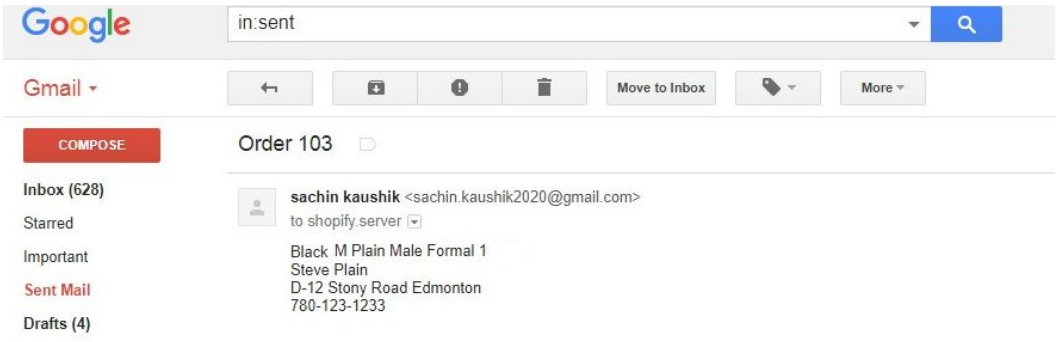


Figure 2: Incomplete or wrong request

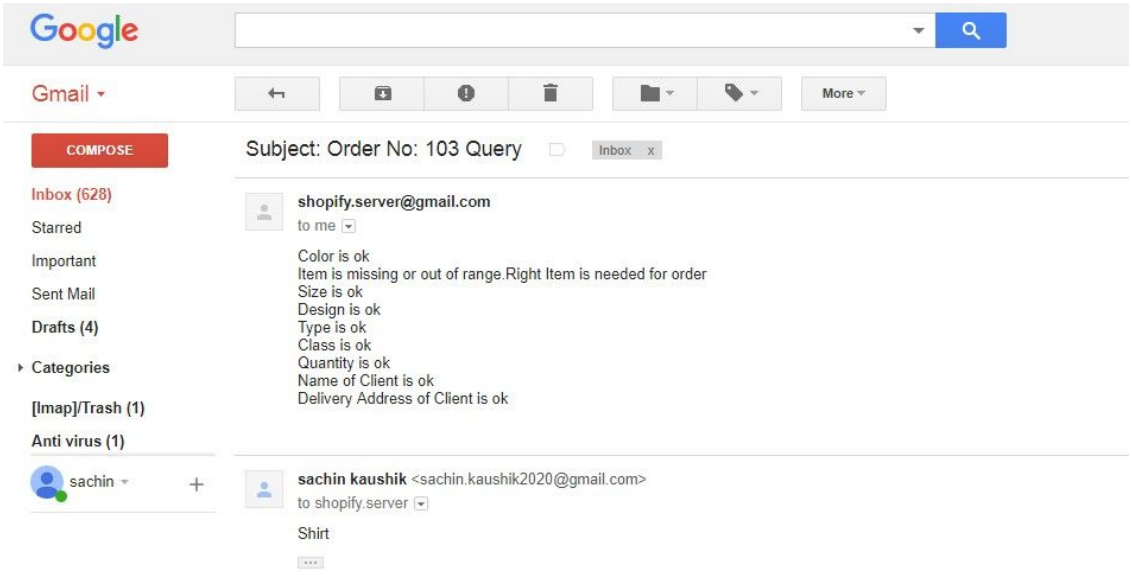


Figure 3: Query from Shopify and reply of client

When confirmed type for shirt is received to E2R system, Mail Q&A can complete REST Invocation and finalize the order. To implement REST Invocation, we chose a different environment called Flask which is micro web framework written in Python.

Obviously, there could have multiple interactions between client and Q&A system which need to be organized in specific way so that e-commerce site and shop owner can refer them as per their requirement. Therefore, State storage is responsible to tag each conversation (Query and Response) with reference number to track all interactions and store them i.e. with order number. Consider, many asynchronous interaction between Mail Q&A and Clients may occur at similar time. One person has received query for the right address and other received query for missed parameter such as design (if ordering garments). State Storage will provide specific number to sort them out, e.g. same order number could be utilized.

It is important to note that all E2R model is inspired from client-server model. The principle behind the client-server constraints is the separation of several concerns. Separating the user interface concerns from the data storage concerns improves the portability of the user interface across multiple platforms.

It also improves scalability by simplifying the server components as we can increase the capacity of clients and servers separately (by adding new nodes to the network). Perhaps most significant to the Web, is that the separation allows the components to evolve independently, thus supporting the Internet-scale requirement of multiple organizational domains. For example, client-server model allows the server and client to be implemented in different programming languages. Design of a client-server application enables that application to be fault-tolerant.

In a fault-tolerant system, failures may occur without causing a shutdown of the entire application. In a fault-tolerant client-server application, one or more servers may fail without

stopping the whole system as long as the services offered on the failed servers are available on servers that are still active. Another advantage of modularity is that a client/server application can respond automatically to increasing or decreasing system loads by adding or shutting down one or more services or servers.

2. Architecture of E2R system

To understand the working structural components, we should know the insights of the architecture of the proposed E2R system (Figure 4).

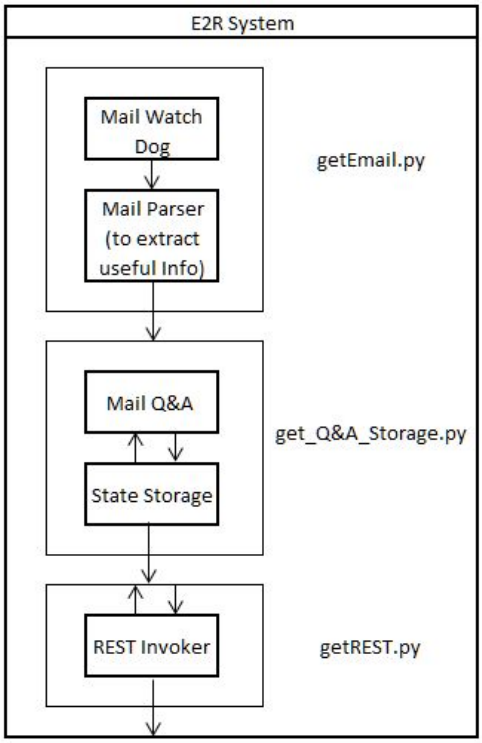


Figure 4: Architecture of E2R System

E2R system is implemented separately from e-commerce server. The E2R architecture has several structural components. Every component has their predefined work which are described below in brief;

1. Mail Watchdog: A simple system to monitor an email box and respond when a new email is received. Depending on the contents of the email, the Watchdog invokes an appropriate Mail Parser. It comes under getEmail.py (Figure 4).

2. Mail Parser: A program to read a specific email, parse the contents for the most relevant information and passes control to the REST Invoker. It also comes under getEmail.py.

3. Mail Q&A (Question and Answer) system: If the Mail Parser is unclear or has questions about the specific REST invocation (i.e., the email is not as well-formed as required), the Q&A system can respond with an email with the proposed REST invocation, ask questions to clarify any unclear portions, and then finalize the REST invocation. Here, Mail Q&A comes under get_Q&A_Storage.py (Figure 4).

In this E2R system, I had implemented three cases for Mail Q&A (for non-well-formed requests), which is as follows;

i. If item's details is missing or out of range then Mail Q&A will express the problem to finalize the order and it will ask client to send information
ii. If Information is unclear/missing then Mail Q&A will ask for specific missing Information from client directly.
iii. If Information is correct and item is available then Mail Q&A will pars the Information and store it permanently in the storage system. Further, Mail Q&A would finalize the REST Invocation.

3. State Storage: Simple storage system to remember the relationship between requests and responses, since there may be multiple interactions (e.g., Mail Q&A system). State Storage System will also store the client information such as Name, Address, delivery address, email ID and phone number etc. Each client would have an assigned unique order ID. It also comes under get_Q&A_Storage.py.

4. REST Invoker: Once the REST invocation is complete, the Invoker actually makes the REST calls, waits/monitors the response, and sends the response email. REST Invoker is implemented separately under getREST.py (Figure 4).

3. Main Working components

There are several working components of E2R system (Figure 5). Yellow color block denotes Client or Shopify (e-commerce site) platform, purple blocks are for E2R system and orange blocks show feedback blocks. Also, green comments are positive comments, red comments and black comments are neutral comments.

However, in this Capstone project, I worked mainly working on Mail Q&A and State Storage system, which are giving more advance feature to E2R system to operate automatically without interventions. The implementation of this E2R system will require the development of a client-server system that incorporates the Simple Mail Transfer Protocol (SMTP), parsing algorithm.

The Mail Parser will extract the email and translate human language to useful data (text). Mail Q&A will generate specific query to client in text format to finalize REST Invocation. Client will reply to E2R with solicited information which will follow the same path like through Mail Watchdog then Mail Parser (text to JSON). State Storage will maintain reference number to track all the asynchronous conversation between Mail Q&A and client.

An invoker will use these data to format and make a REST call to a server which will actually respond to client's requests. After processing requests, the third-party server responds to the E2R System in computer language. At this time, the E2R system will do translation again (JSON to MIME) and send confirmation back to clients via email in natural language.

Technically, the E2R system comes under RESTful Web Architecture. The main programming Language of the E2R system is Python 3. Based on SMTP the E2R system sends and retrieves emails. During this process, DNS and mail header helps to choose right functions to parse emails. Query and Response session is maintained between for Mail Q&A. This interaction is tracked using reference counter in storage system. When reacting with third-party servers, requests, response and session establishment are based on HTTP and REST.

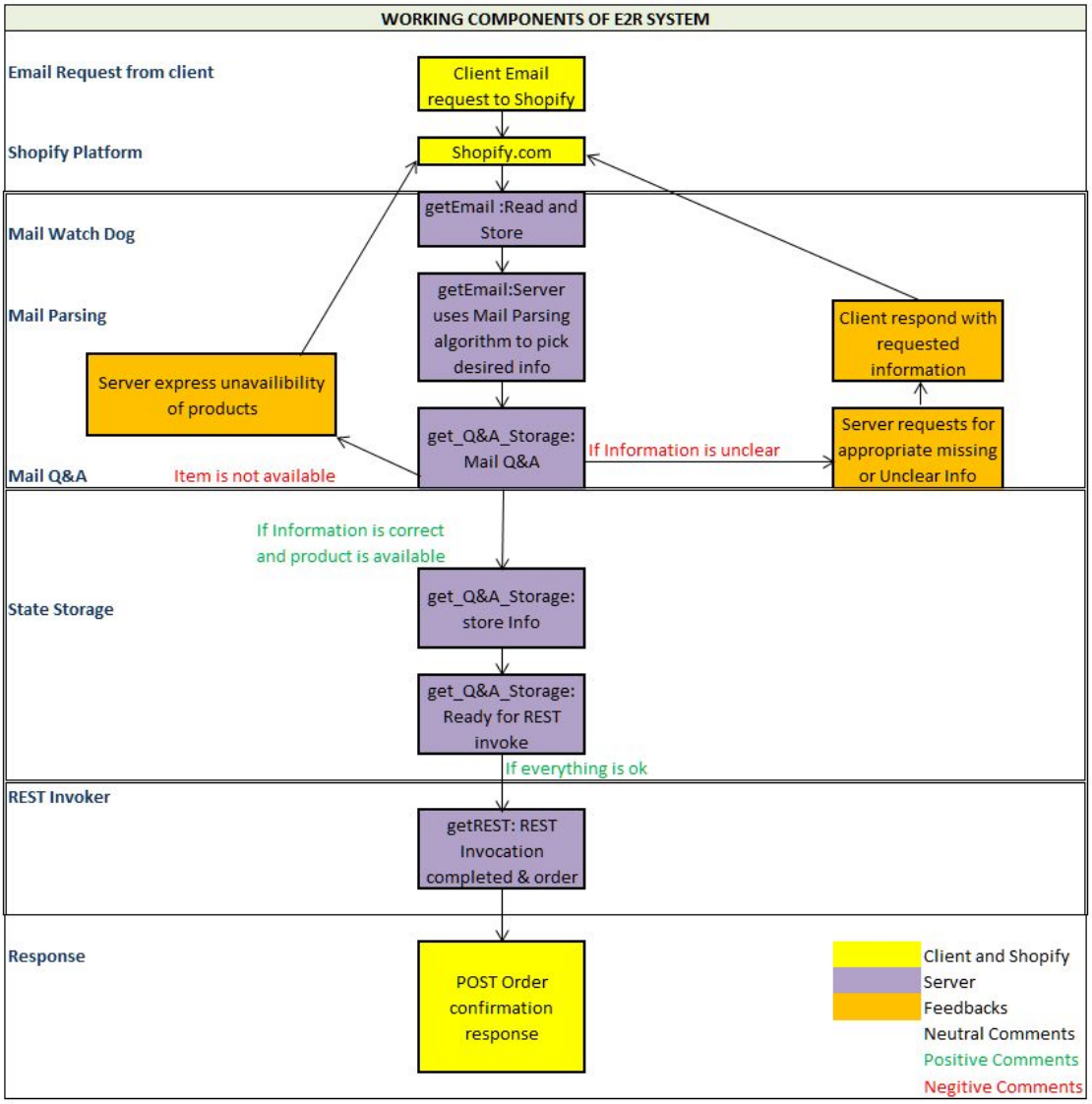


Figure 5: Working components of E2R system

4. Implementation

We have seen that E2R system has several components and these components are necessary for implementation of a fully functional E2R system. However, many different implementations for E2R system is possible depending upon the uses such as with or without of State storage, Security and Mail Q&A along with Watchdog, Mail Parser and REST Invoker. During the implementation of my main working components, it is better to have understanding of each component of the E2R system separately as per execution point of view.

a. Implementation Overview

To execute this project, I worked on tools such as Python3 and made a github student account for coding purposes. Python is an interpreted high-level programming language for general-purpose programming. Python has a design philosophy that emphasizes code readability, and a syntax that allows programmers to express concepts in fewer lines of code, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

GitHub is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

GitHub offers plans for both private repositories and free accounts which are commonly used to host open-source software projects.

During initial execution of the E2R system, first major challenge was to implement basic back bone of operational E2R system which comprises Mail Watchdog, Mail Parser and REST Invoke system.

All idea for implementation was to make a client server model using E2R server and Eureka Shop server. To implement REST Invocation, we chose a different environment called Flask.

Flask is a micro web framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine. It is BSD licensed. Flask is called a micro framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were

implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program.

Whenever E2R server (acting like a client) gets order from client, he just invokes an action called REST to place the order to REST Server (Shop from getting order) with final notification of order placement. Representational State Transfer (REST) or RESTful, allows the requesting systems to access and manipulate textual representations of resources by using a uniform and predefined services of stateless operations. In RESTful web service, requests made to a resources will elicit a response that may be in XML, HTML, JSON or some other format. The operations can be done with GET, PUT, POST & DELETE.

The **REST** architecture was originally designed to fit the HTTP protocol that the world wide web uses. The HTTP request methods are typically designed to affect a given resource in the standard ways.

GET- Obtains information about a resource

POST-Create a new resource

PUT-Update a resource

DELETE-Delete a resource

Flask server script (get_Q&A_Storage.py and getREST.py) has two methods Get and Post. Get is to serve the request for get information and post to post the information on the REST server.

b. Working of each components with results

Working of each component of E2R system is unique and interdependent. We have already discussed about the each component in Figure 1. “Simple architecture of E2R system”. Please see section 2 Architecture of E2R system. Taking about the code of the email Watchdog three things are important to mention. First of all, we need a mail server, a username and password. In this project, since we are using to login with Gmail, our mail server would be either imap.gmail.com or smtp.gmail.com., if one is trying to read the incoming mail server would be imap.gmail.com with port number 993 and if we are trying to send mail then outgoing mail server would be smtp.gmail.com having port number 587. I have created a Shopify ID on Gmail i.e. Shopify.server@gmail.com and a client ID as sachin.kaushik2020@gmail.com.

```
10  #Login into shopify Server
11  From='shopify.server@gmail.com' #http://www.vineetdhanawat.com/blog/2012/06/how-to-extract-email-gmail-contents-as-text
12  Pass='Sachin@123'
13
14  mail=imaplib.IMAP4_SSL("imap.gmail.com",993)
15
16  try:
17      mail.login (From, Pass)
18      print ("LOGIN IS SUCCESSFULL")
19  except imaplib.IMAP4.error:
20      print("LOGIN FAILED!!! ")
21
22  #Select Inbox in email
23  rv, mailboxes = mail.list()
24  if rv == 'OK':
25      print ("Mailboxes:")
26  rv, data = mail.select('inbox')
27  if rv == 'OK':
28      print ("Processing mailbox...\n")
```

Figure 6: Login to Shopify Server (getEmail.py)

In Figure 6: Login to Shopify Server, we have defined our required variables for reading email from Gmail server. I have defined the username and password as From and Pass (line 11 and 12) using which E2R watchdog shall be reading email from established connection and the IMAP server address and port number (line 14). If credentials are correct, login will be successful otherwise there would be a message that "Login is failed" (line 20).

Now we are into the Gmail server. We select inbox. If everything is ok, "Processing mailbox…" (Figure 7) message will print.

```
LOGIN IS SUCCESSFULL
Mailboxes:
Processing mailbox...
```

Figure 7: Output of login into Shopify server (getEmail.py)

Further, our motive to read mails from the Shopify server therefore we start to read the mails from the inbox. Here, we shall search inbox for all mail with search function. We use the built in keyword "ALL" to get all results (documented in RFC3501) (Figure 8: Select Inbox, line 30). We are now going to extract the data we need from the response, then fetch the mail via the ID we just received. It is important to note that imap search function returns a sequential id, meaning id 5 is the 5th email in your inbox.

Here, it indicates that if a E2R server accidentally, deletes emails above email 10 are now pointing to the wrong email. Obviously, this is unacceptable. To resolve this issue, we can ask the imap server to return UID instead. Therefore we use the UID function, and pass in the string of the command in as first argument. The rest behaves exactly the same.

```
22  #Select Inbox in email
23  rv, mailboxes = mail.list()
24  if rv == 'OK':
25      print ("Mailboxes:")
26  rv, data = mail.select('inbox')
27  if rv == 'OK':
28      print ("Processing mailbox...\n")
29
30  rv, data = mail.uid("search", None, "ALL") # Serach all inbox emails
31  # search and return uid instead
32  i=len(data[0].split())#https://gist.github.com/robulouski/7441883
```

Figure 8: Select Inbox (getEmail.py)

Next step is to parsing the raw emails from the E2R server. As we all know the emails in general pretty much look like gibberish and it is a daunting task to extract the meaningful data or message from an email. But, Python has library called email. It can convert raw emails into the familiar email message object. As indicated in the Figure 9: Reading emails from inbox, Fetch is special command function that include the entire email body, or any combination of results such as email flags or Gmail specific IDs such as thread ids (line 39).

We used UTF-8 (Unicode) decoding method to decode emails (line 41). UTF-8 is an encoding schema, like ASCII which is represented with bytes. The difference is that the UTF-8 encoding can be represented every Unicode character, while the ASCII encoding can not. But they are both still bytes. By contrast, an object of type<Unicode> is just that-a Unicode object. After that, we parsed the emails into the relatively useful contents such as details email from, to, subject and body.

```
35  #Reading all emails
36  for x in range (i): #https://pythonprogramminglanguage.com/read-gmail-using-python/
37      pp=int(pp)
38      latest_email_uid=data[0].split()[x]
39      rv, email_data = mail.uid('fetch', latest_email_uid, '(RFC822)')
40      rv_email = email_data[0][1]# Reading email content horizontal and vertical
41      rv_email_string = rv_email.decode('utf-8')
42      # converts byte literal to string removing b'
43      email_message = email.message_from_string(rv_email_string)
44      print('-------------------------------------------------')
45      print ('Email', x)
46
47      aa=email_message ['To']
48
49      bb=email_message ['From']
50
51      ss=email_message ['Subject']
52
53
54
55      if 'Q' not in ss:##Checking if order is new or old
56          pp=str(pp+1)##Providing order number
57          cc='Order No:'+ '+pp
58      else:
59          ss1=ss.split(' ')
60          del ss1[3]
61
62          cc=' '.join(ss1) #https://stackoverflow.com/questions/12453580/concatenate-item-in-list-to-strings
63
64
65      dd=email_message ['Date']
```

Figure 9: Reading emails from Inbox (getEmail.py)

From Figure 10, for loop is defined to check if body is again a byte for all emails in the inbox. If body is not byte than further (line 69), if part checks content types. It will ignore attachments and html. If it is plain text, content of email will be saved into the file using UTF-8 decode format (line 75).

```
68      for part in email_message.walk():# Function to store content of parsed email into the file
69          if part.get_content_type() == "text/plain": # ignore attachments/html
70              body = part.get_payload(decode=True)
71              save_string = "email" + ".txt"
72              # location on disk
73              myfile = open(save_string, 'w')
74              myfile.write("To: %s\n\nFrom: %s\n\nSubject: %s\n\nDate: %s\n\n" %(aa,bb,cc,dd))
75              myfile.write(body.decode('utf-8'))
76
77              myfile.close()
78              file=open("email.txt","r")
79              print (file.read())
80
81          else:
82              continue
83
84              # body is again a byte literal
85              myfile.close()
86
87      # this will loop through all the available multiparts in mail
```

Figure 10: Storing emails into file (getEmail.py)

Afterwards, file is opened and we print the contents of saved emails as shown in the Figure 11.

```
==== RESTART: C:\Users\Sachin Kaushik\capstone.git\E2R_System\getEmail.py ====
LOGIN IS SUCCESSFULL
Mailboxes:
Processing mailbox...

---------------------------------------------------
Email 0
To: shopify.server@gmail.com

From: sachin kaushik <sachin.kaushik2020@gmail.com>

Subject: Order No: 101

Date: Tue, 27 Feb 2018 21:45:48 -0700

Black Pant M Plain Male Formal 1

Steve Plain

D-12 Stony Road Edmonton

780-123-1233
```

Figure 11: Email output at getEmail.py

Content of email is sent to Flask get Q&A and Storage server Figure 12. Here, we used post request with HTTP connection over 127.0.0.1 and port 2000 (line 93).

```
91      print ("------------------New Order Notification----------------")
92      #Posting content of New order to Q&A and Storage Server
93      r=requests.post('http://127.0.0.1:2000/server/storage', json=(lines))
94      print (r.reason)
```

Figure 12: Posting order to Q&A and Storage Server

During the implementation of the Q&A portion of E2R, we created three most possible scenarios as discussed in the Section 2: Architecture of E2R System. In the very first step, we created lists which will be used as database Figure 13.

```
#Creating various dictionaries. Used as running database/memory
Order_ID=[]
Item=['Pant','Shirt','Sweater','Coat']
Type=['Male','Female']
Color=['Blue','Red','Black','White']
Size=['S','M','L','XL']
Design=['Plain','Strips']
Class=['Casual','Formal']
Quantity=['1','2']
Client_Name=[]
Client_Delivery_Address=[]
Client_Phone_Number=[]
From='shopify.server@gmail.com'
Pass='Sachin@123'
```

Figure 13: List as database (get_Q&A_Storage.py)

Being Flask server, getEmail.py has sent the parsed contents to Q&A and Storage. This server is running on 127.0.0.1 and port 2000. All contents are first converted to JSON to string then string to lists. Further, strip function is additionally used to rectify the possibility of unintended '\n' into the result (Figure 14 line 51,52,53,54 & 55). Split function will now convert string into lists (line 57, 59, 61, 63 & 65).

```
#Creating server route with method GET, POST
@app.route('/server/storage', methods=['GET','POST'])#https://www.youtube.com/watch?v=CjYKrbq8BCw&t=21s&list=PLiaVlvtF4yDucFTMgrGRHWhAkL9QLG1vd&index=14
def raw_order():
        try:
                if request.get_json():
                        content=request.get_json()#Conversted Json content into the string
                        pattern=content[0]#Select the each string
                        pattern1=content[2]
                        pattern2=content[4]
                        pattern3=content[8]
                        pattern4=content[10]
                        pattern5=content[12]
                        pattern6=content [14]

                        pattern3s=pattern3.strip('\n')#Strip unnecessary '\n' from each string
                        pattern4s=pattern4.strip('\n')
                        pattern5s=pattern5.strip('\n')
                        pattern6s=pattern6.strip('\n')
                        pattern2s=pattern2.strip('\n')

                        work=pattern3s.split(' ')#Now converting each string into lists
                        print (work)
                        work1=pattern4s.split(' ')
                        print (work1)
                        work2=pattern5s.split(' ')
                        print (work2)
                        work3=pattern6s.split(' ')
                        print (work3)
                        work4=pattern2s.split(' ')
                        print (work4)
```

Figure 14: Converting to lists (get_Q&A_Storage.py)

Case 1: During first case, all the list contents will be tried to print. In case of any error, when strings are empty command takes to our first case of Q&A. In that case, some clarification is received.

```
#Try Statement to print requested order. If details exceeds, command goes to except and completes first part of Q&A
try:
        aa='Requested Order details:'+work[0]+' '+work[1]+' '+work[2]+' '+work[3]+' '+work[4]+' '+work[5]+' '+work[6]
        print (aa)
        bb='Name of client:'+work1[0]+' '+work1[1]
        print (bb)
        for i in range (len(Client_Name)):
                if Client_Name[i]==work1[i]:
                        print('Client name into database is upto date')
                else:
                        Client_Name.append(work1)#https://www.tutorialspoint.com/python/list_append.html
                        print('Client name is updated into database')

        cc='Delivery address of client:'+work2[0]+' '+work2[1]+' '+work2[2]+' '+work2[3]
        print (cc)
        for i in range (len(Client_Delivery_Address)):
                if Client_Delivery_Address[i]==work2[i]:
                        print('Client delivery address is upto date into database ')
                else:
                        Client_Delivery_Address.append(work2)
                        print('Client Address is updated into database')
        dd='Phone number of client:'+work3[0]
        print (dd)
        for i in range (len(Client_Phone_Number)):
                if Client_Phone_Number[i]==work3[i]:
                        print('Client phone number is upto date')
                else:
                        Client_Phone_Number.append(work3)
                        print('Client phone number is updated')
```

Figure 15: Case 1: Q&A (get_Q&A_Storage.py)

During the same step, we are also updating database in form of list. If there is any extra or less information is provided by the client then error will be handled and a query will be placed to client. Please see code in Figure 15 and Figure 16.

```
except:
        wq='Order details are either missing or incorrect'
        print (wq)

        g='Requested order details are incorrect. Please enter details as follows:\n1.Color of Item\n2.Item Name\n3.Size of Item\n4.Design of Item
        wa='Query has been successfully placed to the current client for'+' '+pattern2s

        msg=MIMEMultipart() ##https://docs.python.org/2/library/email-examples.html
        msg['from']=From
        msg['To']=pattern1
        msg['Subject']=pattern2+' '+'Q'
        msg.attach(MIMEText(g,'plain'))
        mail=smtplib.SMTP('smtp.gmail.com',587)
        mail.starttls()
        message=msg.as_string()
        mail.login (From, Pass)
        mail.sendmail(pattern, pattern1,message)
        print('Query has been successfully placed to the current client for'+' '+pattern2s)
        mail.close()
        client_query={'Query has been successfully placed to the current client for'+' '+pattern2s}
        return jsonify(wa)
```

Figure 16: Case 1: Exception handling (get_Q&A_Storage.py)

During error handling, if there is any missing or out of range detail found then a message will be printed as shown in Figure 17 and Figure 18. Output on get_Q&A_Storage.py idle is shown in Figure 17 and getEmail.py in Figure 18. The response email for customer is indicated in Figure 19.

```
127.0.0.1 - - [04/Apr/2018 11:33:55] "POST /server/storage HTTP/1.1" 200 -
['Black', 'Plain', 'Male', 'Pant', 'M', '1']
['Steve', 'Plain']
['D-12', 'Stony', 'Road', 'Edmonton']
['780-123-1233']
['Subject:', 'Order', 'No:', '102']
Order details are either missing or incorrect
Query has been successfully placed to the current client for Subject: Order No: 102
```

Figure 17: Case 1- get_Q&A_Storage.py output

```
Query has been successfully placed to the current client for Subject: Order No: 102
Waiting for Clients clarification
```

Figure 18: Case 1-getEmail.py output

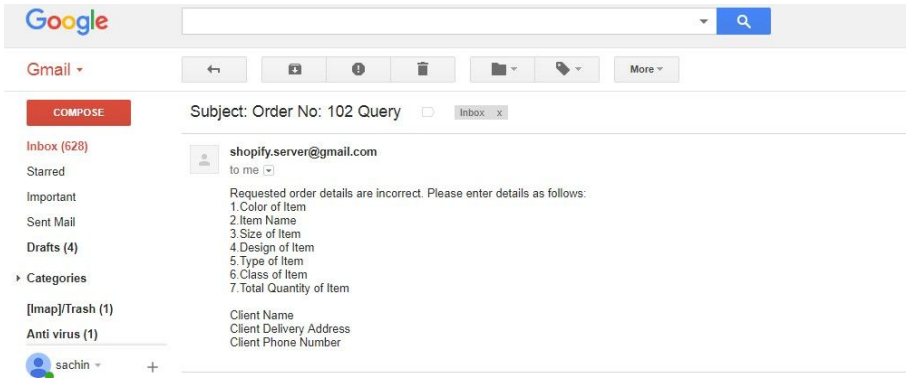
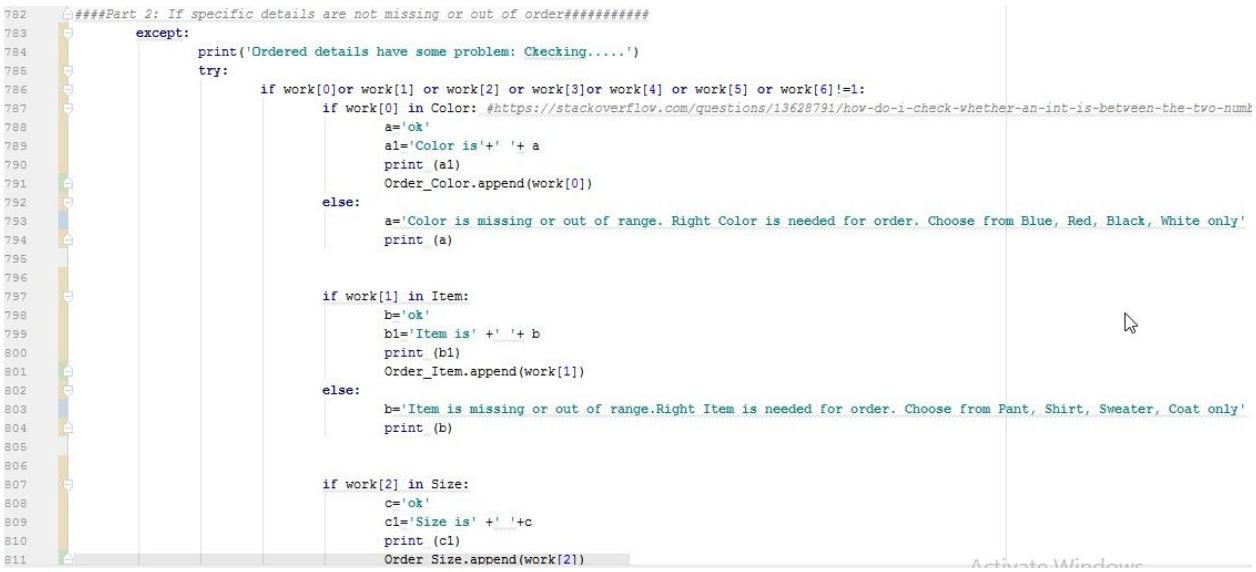


Figure 19: Case 1-Email of customer

In Figure 16, an email is created by passing client email, message (requesting about details) with similar subject. Here, we used SMTP server to send email of Gmail (line 744). We logged into the SMTP server of Google with smtp.gmail.com with port number 587. Message format is into MIME multipart. (line 739). Email parts are taken from the parsed information such as email from, email to and email subject. Message is created as "Query has been placed successfully to the current client". There will be also message posted into shell "Query has been placed successfully to the current client" (Figure 19).

Case 2: If some details are correct, Q&A will check deep into information and find out the specific details to be asked for. Each detail will be checked from the database list and if anything found unsatisfactory an email will be sent asking about specific detail. Please see Figure 20 and Figure 21 portion of code for creating this case. Partial information is stored into list using append function. For example see line 791.

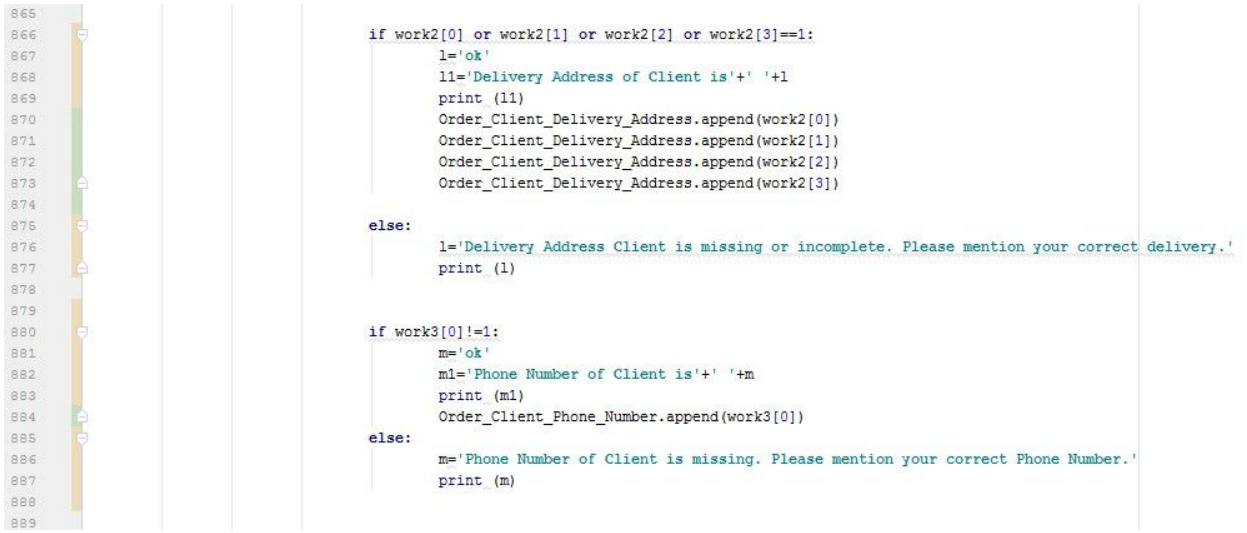


```
####Part 2: If specific details are not missing or out of order###########
        except:
                print('Ordered details have some problem: Checking.....')
                try:
                        if work[0]or work[1] or work[2] or work[3]or work[4] or work[5] or work[6]!=1:
                                if work[0] in Color: #https://stackoverflow.com/questions/13628791/how-do-i-check-whether-an-int-is-between-the-two-numb
                                        a='ok'
                                        a1='Color is'+' '+ a
                                        print (a1)
                                        Order_Color.append(work[0])
                                else:
                                        a='Color is missing or out of range. Right Color is needed for order. Choose from Blue, Red, Black, White only'
                                        print (a)


                                if work[1] in Item:
                                        b='ok'
                                        b1='Item is' +' '+ b
                                        print (b1)
                                        Order_Item.append(work[1])
                                else:
                                        b='Item is missing or out of range.Right Item is needed for order. Choose from Pant, Shirt, Sweater, Coat only'
                                        print (b)


                                if work[2] in Size:
                                        c='ok'
                                        c1='Size is' +' '+c
                                        print (c1)
                                        Order_Size.append(work[2])
```

Figure 20: Case 2-Checking about specific details (get_Q&A_Storage.py)



```
                if work2[0] or work2[1] or work2[2] or work2[3]==1:
                        l='ok'
                        l1='Delivery Address of Client is'+' '+l
                        print (l1)
                        Order_Client_Delivery_Address.append(work2[0])
                        Order_Client_Delivery_Address.append(work2[1])
                        Order_Client_Delivery_Address.append(work2[2])
                        Order_Client_Delivery_Address.append(work2[3])

                else:
                        l='Delivery Address Client is missing or incomplete. Please mention your correct delivery.'
                        print (l)


                if work3[0]!=1:
                        m='ok'
                        m1='Phone Number of Client is'+' '+m
                        print (m1)
                        Order_Client_Phone_Number.append(work3[0])
                else:
                        m='Phone Number of Client is missing. Please mention your correct Phone Number.'
                        print (m)
```

Figure 21: Case 2-Checking about specific details (get_Q&A_Storage.py)

In Figure 22, it is pertinent to note that further if statement is implemented to set the values of variables such as a1,b1 etc. Once required string is passed e.g. if parameter is fine it will be ‘OK’ otherwise 'Quantity is missing or out of range. Right Quantity is needed for order' will be passed (line 892,893,894, 895 etc). Once all the required information see passed into variables then one combined string is made (line 907). Further, this string passed a message during asking about the clarifications from the client (line 918).

```
#Checking status of each parameter of ordered details
        if a!='ok':
                a1='Color is missing or out of range. Right Color is needed for order'
        if b!='ok':
                b1='Item is missing or out of range.Right Item is needed for order'
        if c!='ok':
                c1='Size is missing or out of range.Right Size is needed for order'
        if d!='ok':
                d1='Design is missing or out of range.Right Design is needed for order'
        if e!='ok':
                e1='Type is missing or out of range.Right Type is needed for order'
        if f!='ok':
                f1='Class is missing or out of range.Right Class is needed for order'
        if h!='ok':
                h1='Quantity is missing or out of range.Right Quantity is needed for order'

        content=a1+'\n'+b1+'\n'+c1+'\n'+d1+'\n'+e1+'\n'+f1+'\n'+h1+'\n'+k1+'\n'+l1+'\n'

        msg=MIMEMultipart() ##https://docs.python.org/2/library/email-examples.html
        msg['from']=From
        msg['To']=pattern1
        msg['Subject']=pattern2s+' '+'Q'
        msg.attach(MIMEText(content,'plain'))
        mail=smtplib.SMTP('smtp.gmail.com',587)
        mail.starttls()
        message=msg.as_string()
        mail.login (From, Pass)
        mail.sendmail(pattern, pattern1,message)
        print('Query has been successfully placed to the current client for'+' '+pattern2s)
        mail.close()
        return jsonify('Query has been successfully placed to the current client for'+' '+pattern2s)
```

Figure 22: Case 2: Asking about Specific details (get_Q&A_Storage.py)

Before sending queries all the information is stored into the list as a running memory and we will wait clarification from client (Figure 23).

```
#Store current order and wait for client to repsond
Order_Item=[]
Order_Type=[]
Order_Color=[]
Order_Size=[]
Order_Design=[]
Order_Class=[]
Order_Quantity=[]
Order_Client_Name=[]
Order_Client_Delivery_Address=[]
Order_Client_Phone_Number=[]
```

Figure 23: Case 2: Order details stored (get_Q&A_Storage.py)

Output on get_Q&A_Storage.py idle is shown in Figure 24 and getEmail.py in Figure 25. The response email for customer is indicated in Figure 26.

```
Ordered details have some problem: Ckecking.....
Color is ok
Item is missing or out of range.Right Item is needed for order
Size is missing or out of range.Right Size is needed for order
Design is ok
Type is ok
Class is ok
Quantity is ok
Name of Client is ok
Delivery Address of Client is ok
Phone Number of Client is ok
Query has been successfully placed to the current client for Subject: Order No: 102
```

Figure 24: Case 2- get_Q&A_Storage.py output

```
------------------New Order Notification----------------
OK
Query has been successfully placed to the current client for Subject: Order No: 102
Waiting for Clients clarification
```

Figure 25: Case 2-getEmail.py output

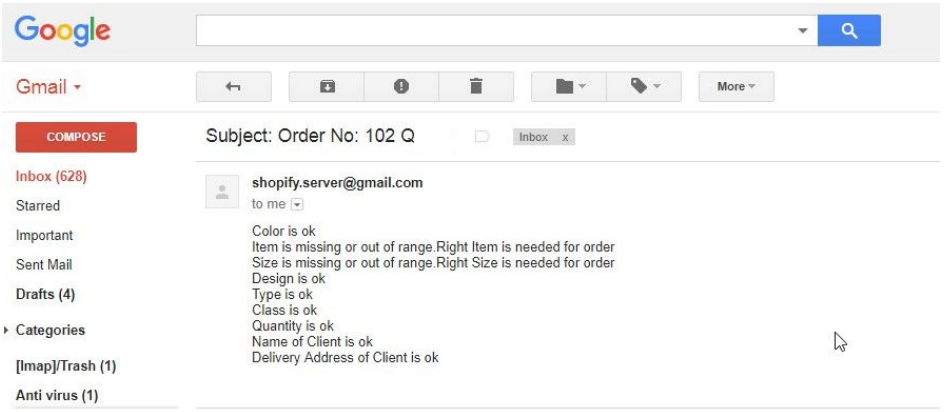


Figure 26: Case 2-Customer's received query

Sending reply: Client now replying with specific information that has been requested. In current case client provide item and size of item (Figure 27).

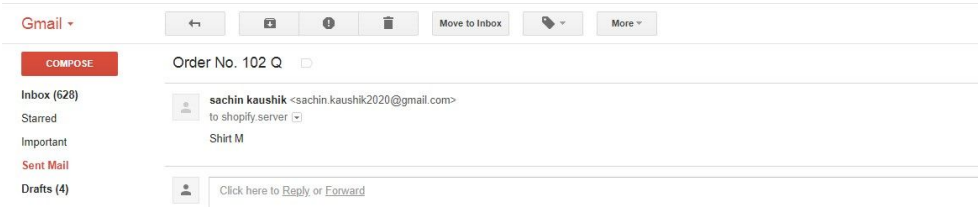


Figure 27: Client's reply

This information is again parsed from getEmail.py and sent to get_Q&A_Storage.py. Information is checked and generated error which is handled by exception handling statement (Figure 28).

Typically, information has to be checked for every possible category therefore we have created nested try and except loop to check field of every category (Figure 29, 30, 31, 32, 33,34). When, we found the desired information that is added into the stored information. Once all the required parameters are received, Q&A and Storage invokes the REST to finalize the order (Figure 35).

```
except:
        content=request.get_json()
        pattern2=content[4]
        pattern2s=pattern2.strip('\n')
        work4=pattern2s.split(' ')
        print ('Clarification is received for'+' '+pattern2s)
        pattern3=content[8]
        pattern3s=pattern3.strip('\n')
        work=pattern3s.split(' ')
        try:
                qs='Requested Order details:'+work[0]+' '+work[1]+' '+work[2]+' '+work[3]+' '+work[4]+' '+work[5]+' '+work[6]
                if work[0] or work[1] or work[2] or work[3] or work[4] or work [5] or work [6] in Item:
                        if work[0] or work[1] or work[2] or work[3] or work[4] or work [5] in Item:
                                if work[0] or work[1] or work[2] or work[3] or work[4] in Item:
                                        if work[0] or work[1] or work[2] or work[3] in Item:
                                                if work[0] or work[1] or work[2] in Item:
                                                        if work[0] or work[1] in Item:
                                                                if work[0] in Item:
                                                                        Order_Item.append(work[0])

                                                                else:
                                                                        Order_Item.append(work[1])
                                                        else:
                                                                Order_Item.append(work[2])
                                                else:
                                                        Order_Item.append(work[3])
                                        else:
                                                Order_Item.append(work[4])
                                else:
                                        Order_Item.append(work[5])
                        else:
                                Order_Item.append(work[6])
```

Figure 28: Checking specific detail from reply (get_Q&A_Storage.py)

```
    except:
            try:
                    qs='Requested Order details:'+work[0]+' '+work[1]+' '+work[2]+' '+work[3]+' '+work[4]+' '+work[5]
                    if work[0] or work[1] or work[2] or work[3] or work[4] or work [5] in Item:
                            if work[0] or work[1] or work[2] or work[3] or work[4] in Item:
                                    if work[0] or work[1] or work[2] or work[3] in Item:
                                            if work[0] or work[1] or work[2] in Item:
                                                    if work[0] or work[1] in Item:
                                                            if work[0] in Item:
                                                                    Order_Item.append(work[0])
                                                            else:
                                                                    Order_Item.append(work[1])
                                                    else:
                                                            Order_Item.append(work[2])
                                            else:
                                                    Order_Item.append(work[3])
                                    else:
                                            Order_Item.append(work[4])
                            else:
                                    Order_Item.append(work[5])
```

Figure 29: Checking specific detail from reply (get_Q&A_Storage.py)

```
            except:
                    try:
                            qs='Requested Order details:'+work[0]+' '+work[1]+' '+work[2]+' '+work[3]+' '+work[4]
                            if work[0] or work[1] or work[2] or work[3] or work[4] in Item:
                                    if work[0] or work[1] or work[2] or work[3] in Item:
                                            if work[0] or work[1] or work[2] in Item:
                                                    if work[0] or work[1] in Item:
                                                            if work[0] in Item:
                                                                    Order_Item.append(work[0])
                                                            else:
                                                                    Order_Item.append(work[1])
                                                    else:
                                                            Order_Item.append(work[2])
                                            else:
                                                    Order_Item.append(work[3])
                                    else:
                                            Order_Item.append(work[4])
```

Figure 30: Checking specific detail from client's reply (get_Q&A_Storage.py)

```
except:
        try:
                qs='Requested Order details:'+work[0]+' '+work[1]+' '+work[2]+' '+work[3]
                if work[0] or work[1] or work[2] or work[3] in Item:
                        if work[0] or work[1] or work[2] in Item:
                                if work[0] or work[1] in Item:
                                        if work[0] in Item:
                                                Order_Item.append(work[0])
                                        else:
                                                Order_Item.append(work[1])
                                else:
                                        Order_Item.append(work[2])
                        else:
                                Order_Item.append(work[3])
```

Figure 31: Checking specific detail from client's reply (get_Q&A_Storage.py)

```
except:
        try:
                qs='Requested Order details:'+work[0]+' '+work[1]+' '+work[2]
                if work[0] or work[1] or work[2] in Item:
                        if work[0] or work[1] in Item:
                                if work[0] in Item:
                                        Order_Item.append(work[0])
                                else:
                                        Order_Item.append(work[1])
                        else:
                                Order_Item.append(work[2])
```

Figure 32: Checking specific detail from client's reply (get_Q&A_Storage.py)

```
except:
        try:
                qs='Requested Order details:'+work[0]+' '+work[1]
                if work[0] or work[1] in Item:
                        if work[0] in Item:
                                Order_Item.append(work[0])
                        else:
                                Order_Item.append(work[1])
```

Figure 33: Checking specific detail from client's reply (get_Q&A_Storage.py)

```
except:
        try:
                qs='Requested Order details:'+work[0]
                if work[0] in Item:
                        Order_Item.append(work[0])
                if work[0] in Type:
                        Order_Type.append(work[0])
                if work[0] in Color:
                        Order_Color.append(work[0])
                if work[0] in Size:
                        Order_Size.append(work[0])
                if work[0] in Design:
                        Order_Design.append(work[0])
                if work[0] in Class:
                        Order_Class.append(work[0])
                if work[0] in Quantity:
                        Order_Quantity.append(work[0])
```

Figure 34: Checking specific detail from client's reply (get_Q&A_Storage.py)

Case 3: If every details are correct and matched with the list created into the Q&A, else statement will indicate the message that every details is correct and confirmed client details of order will be shown once again as shown in the Figure 35.

```
690
691             print('Order details are correct')
692             client_order={'Order is confirmed for':pattern2s,'Color': Order_Color,
693             'Item': Order_Item,'Size' : Order_Size, 'Design': Order_Design, 'Type': Order_Type,
694             'Class': Order_Class, 'Quantity': Order_Quantity,'Name of client': Order_Client_Name,
695             'Delivery address of client':Order_Client_Delivery_Address,
696             'Phone number of client':Order_Client_Phone_Number}
697             return jsonify(client_order)
```

Figure 35: If every detail is correct. Place order. (get_Q&A_Storage.py)

At getEmail.py final details will be posted to REST server Figure 36. Server is connected through HTTP connection via IP 127.0.0.1 and port 5000. Content is sent in JSON format (line 99).

```
91  print ("------------------New Order Notification----------------")
92  #Posting content of New order to Q&A and Storage Server
93  r=requests.post('http://127.0.0.1:2000/server/storage', json=(lines))
94  print (r.reason)
95  content=r.json()
96  print (content) # Print the return content either confirmation of order or placed query
97
98  if 'Query' not in content:#Condition if notification is not Query then proceed with final order confirmation
99      req=requests.post('http://127.0.0.2:5000/server',json=(content))#Post order details to REST Server
100     content1=req.json()
101     print (content1)
102 else:
103     print('Waiting for Clients clarification')
104
105 mail.close()
106
```

Figure 36: Final REST Invocation (getEmail.py)

Output on get_Q&A_Storage.py idle is shown in Figure 37 and getEmail.py idle in Figure 38.

```
 * Running on http://127.0.0.1:2000/ (Press CTRL+C to quit)
['Black', 'Pant', 'M', 'Plain', 'Male', 'Formal', '1']
['Steve', 'Plain']
['D-12', 'Stony', 'Road', 'Edmonton']
['780-123-1233']
['Subject:', 'Order', 'No:', '101']
Requested Order details:Black Pant M Plain Male Formal 1
Name of client:Steve Plain
Delivery address of client:D-12 Stony Road Edmonton
Phone number of client:780-123-1233
127.0.0.1 - - [04/Apr/2018 11:13:36] "POST /server/storage HTTP/1.1" 200 -
```

Figure 37: Case 3- get Q&A_Storage.py output

```
------------------New Order Notification----------------
OK
{'Class': 'Formal', 'Color': 'Black', 'Delivery address of client': ['D-12', '
Stony', 'Road', 'Edmonton'], 'Design': 'Plain', 'Item': 'Pant', 'Name of clien
t': ['Steve', 'Plain'], 'Order Number': 'Delivery address of client:D-12 Stony
Road Edmonton', 'Order is confirmed for': 'Subject: Order No: 101', 'Phone num
ber of client': '780-123-1233', 'Quantity': '1', 'Size': 'M', 'Type': 'Male'}
New Order is posted
```

Figure 38: Case 3-getEmail.py output

In the last part of Q&A, if there is occurrence of any Index error there must be some problem with data client has entered. Therefore, E2R server will have to place query asking for correct details as shown in the Figure 39. Similarly, this part has to place query by sending emails to current client as we used above part of Q&A.

```
925             except IndexError: #If any error occurred, Q&A will ask for more details:
926                     g='Requested order details are not in correct order. Please enter details as follows:\n1.Color of Item\n2.Item Name\n3.Size of Ite
927                     print (g)
928
929                     msg=MIMEMultipart() ##https://docs.python.org/2/library/email-examples.html
930                     msg['from']=From
931                     msg['To']=pattern1
932                     msg['Subject']=pattern2s+' '+'Q'
933                     msg.attach(MIMEText(g,'plain'))
934                     mail=smtplib.SMTP('smtp.gmail.com',587)
935                     mail.starttls()
936                     message=msg.as_string()
937                     mail.login (From, Pass)
938                     mail.sendmail(pattern, pattern1,message)
939                     print('Query has been successfully placed to the current client for'+' '+pattern2s)
940                     mail.close()
941
942
943     return app
```

Figure 39: For any type of error (get_Q&A_Storage.py)

Now, we will establish an HTTP connection with the Eureka shop server Figure 40. Here, we are using just post function of RESTful services. Connection of the Eureka shop server are maintained as a local host server which runs on 127.0.0.2 IP address and 5000 port.

```
...
============ RESTART: C:\Users\Sachin Kaushik\Desktop\getREST.py ============
 * Running on http://127.0.0.2:5000/ (Press CTRL+C to quit)
```

Figure 40: Running REST server (getREST.py)

A separate script of REST call to accept the order will be running which just accept the order details and paste into their result as shown in the Figure 41 and Figure 42. In the REST, we are operating through GET and POST methods. We defined if and else statement to check format of received data from getEmail.py. If data is received then confirmed ordered content will be posted (line 11) otherwise no order will be printed (line 13).

```
1   from flask import Flask, json, request, jsonify
2
3   app =Flask(__name__)
4
5   @app.route('/server', methods=['GET','POST'])#https://www.youtube.com/watch?v=lUCmVNGs5gw&t=0s&list=PLiaVlvtF4yDucFTMgrGRHWhAkL9QLG1vd&index=10
6   def neworder(): ## https://techtutorialsx.com/2017/01/08/esp8266-posting-json-data-to-a-flask-server-on-the-cloud/
7           if request.get_json():
8                   content=request.get_json()#Get contents from Json format
9                   print ('\n-----New Order details as follows----\n\n')
10                  print (content)
11                  return jsonify('New Order is posted')# Post New order Notification
12          else:
13                  return "No new order" #return "no order"
14  if __name__=='__main__':
15          app.run(host='127.0.0.2', port=5000)
16          app.debug=True
17
```

Figure 41: REST Server code (getREST.py)

```
 * Running on http://127.0.0.2:5000/ (Press CTRL+C to quit)

-----New Order details as follows----


{'Class': 'Formal', 'Color': 'Black', 'Delivery address of client': ['D-12', 'St
ony', 'Road', 'Edmonton'], 'Design': 'Plain', 'Item': 'Pant', 'Name of client':
['Steve', 'Plain'], 'Order Number': 'Delivery address of client:D-12 Stony Road
Edmonton', 'Order is confirmed for': 'Subject: Order No: 101', 'Phone number of
client': '780-123-1233', 'Quantity': '1', 'Size': 'M', 'Type': 'Male'}
127.0.0.1 - - [04/Apr/2018 13:57:35] "POST /server HTTP/1.1" 200 -
```

Figure 42: Final Order confirmation (getREST.py)

4. How this capstone project is unique

I declare that I have not used any portion of code from Wenting any other student who worked on the other E2R system. I have cited all the URL reference, I used in my code at Appendix A. Further, I have acknowledged the used online resources in all .py files in form of comments. (getEmail.py, get_Q&A_Storage.py and getREST.py)

5. Conclusion

RESTful services using API are very popular on web based application now-a-days. E2R model is one of the best suitable server model can be opt to design an online order system. We support for E2R model for several reasons. First, it is easy to implement by using well-formed emails from client. It can handle various types of requests depending on the type of service we want to provide for example whether it is Q&A service, state storage service.

We have learned several things from this project. First is the use of RESTful API service and its uses. RESTful services are one of the key of this project. After that, we can say parsing is also very important phenomenon to extract desired data from any raw supply of information. We learned how to extract the useful data.

Designing a desired algorithm is way to tackle parsing problems. In last but not least, state storage, this is one of the main components of the implemented E2R system because as we discussed all the information have to be stored and processed by E2R which was a challenging part in itself. However, we took simplest way to implement state storage. We formed an REST server to invoke decisions and create data into lists. This is one of the ways to implement. We can also implement it using Python SQL libraries. But this will make things more complex so we tried to make it simple.

Appendix:B- REFERENCES

[1] https://mailparser.io/i/email-to-rest-api-webhook
[2] https://www.juliedesk.com
[3] https://docparser.com/
[4] http://shop.oreilly.com/product/0636920021575.do
[5] https://en.wikipedia.org/wiki/Representational_state_transfer
[6] http://www.oracle.com/technetwork/articles/java/json-1973242.html
[7] https://docs.python.org/2/library/httplib.html
[8] https://stackoverflow.com/questions/12806386/standard-json-api-response-format
[9] https://www.copterlabs.com/json-what-it-is-how-it-works-how-to-use-it/
[10] https://stackoverflow.com/questions/17301938/making-a-request-to-a-restful-api-using-python
[11] https://en.wikipedia.org/wiki/Python_(programming_language)
[12] https://en.wikipedia.org/wiki/Flask_(web_framework)
[13] https://en.wikipedia.org/wiki/GitHub
[14] https://en.wikipedia.org/wiki/GitHub
[15] https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask
[16] https://auth0.com/blog/developing-restful-apis-with-python-and-flask/
[17] http://www.vineetdhanawat.com/blog/2012/06/how-to-extract-email-gmail-contents-as-text-using-imaplib-via-imap-in-python-3/
[18] https://gist.github.com/robulouski/7441883
[19] https://pythonprogramminglanguage.com/read-gmail-using-python/
[20] https://stackoverflow.com/questions/12453580/concatenate-item-in-list-to-strings
[21] https://www.youtube.com/watch?v=efpFDaXOG6Y&t=466s
[22]https://www.youtube.com/watch?v=CjYKrbq8BCw&t=21s&list=PLiaVlvtF4yDucFTMgrGRHWhAkL9QLG1vd&index=14
[23] https://codereview.stackexchange.com/questions/44168/streamlined-for-loop-for-comparing-two-lists
[24] https://www.tutorialspoint.com/python/list_append.html
[25] https://docs.python.org/2/library/email-examples.html
[26] https://stackoverflow.com/questions/13628791/how-do-i-check-whether-an-int-is-between-the-two-numbers
[27] https://docs.python.org/2/library/email-examples.html
[28] https://www.tutorialspoint.com/restful/index.htm
[29]https://en.wikipedia.org/wiki/Client%E2%80%93server_model
[30https://docs.oracle.com/cd/E13203_01/tuxedo/tux80/atmi/intbas3.htm