

MINT Capstone Project

Final Report

Email-to-REST System(E2R)

User-Request Translation System

Instructor-PROF. PAUL LU

Student- NAVDEEP KAUR

Date: March 13, 2018

Acknowledgement

I would like to express my special thanks to Prof. Paul Lu for providing me his supervision for the Capstone Project. This project is based on the E2R System which is focused mainly on the Main Question and Answer (Q and A) System component of the E2R System.

Introduction

The Email-to-Rest System (referred to as E2R system) is based on developing an automatic user-request translation system. In the E2R system, when a new well-formed email request is received by an email receiver, it results in an invocation to a server using REST (Representational State Transfer). The response of server is also in the form of a well-formed email. The capstone project is based on developing an E2R system (Figure 1) in which the client takes the email requests from the customer to places the order, read and parses the contents of email, and uses the REST invocation to the Main Server to enter the order into a computer system by automatic process, without the need of human intervention, that helps to save time for manual check, read and write information and this results in a gain of productivity.



Figure 1 Basic diagram of the E2R System

The E2R User-Request Translation System is inspired from the popular e-Commerce site Shopify, mailing lists like Google News Alerts, and email based virtual assistants Juliedesk and x.ai which takes the emails, parses emails and takes an appropriate action (e.g., parses the useful information from email and fix meetings for x.ai or juliedesk.com) automatically on server.

Overview and Implementation of the E2R System

The basic idea to design the E2R components (Figure 2) such as Mail Watchdog, Mail Parser, Mail Question and Answer (Q and A) System and the REST invoker is to automate the process by taking an email, parsing it, and use REST calls automatically. But, if the Mail Parser parses contents that has not complete relative information or has some missing arguments in email, then the Mail Q and A system [(C) in Figure 2] responds through an email to customer from the client with the proposed REST invocation, ask questions to customer for clarification of any incomplete information, before finalizing the REST invocation to the Main Warehouse Server.

The MINT Capstone Project is focused on implementation of the Mail Question and Answer (Q and A) component (Figure 2) of the E2R System. It improves the quality and optimization of online shop, as it allows to ask questions by sending an email from administrator of online shop to the customer about the ordered item.

It helps the client (Jaya's online shop) to get clarification about details of order from the customer if any information is incomplete before finalization of REST Invocation to the Main Warehouse REST Server and placement of order. As Question and Answer system is a State-Server between the Main Warehouse Server and the client, therefore, it makes a REST call automatically to the client for clarified details when incomplete information is passed through it and responds only to the Main Server when complete information is given by the client.

Therefore, the simple implementation of the E2R passes through four different stages to develop a client-server architecture that incorporates Mail Watchdog (A), Mail Parser (B), Mail Q and A System (C) and final Main REST Invocation Server (D) as shown in Figure 2.

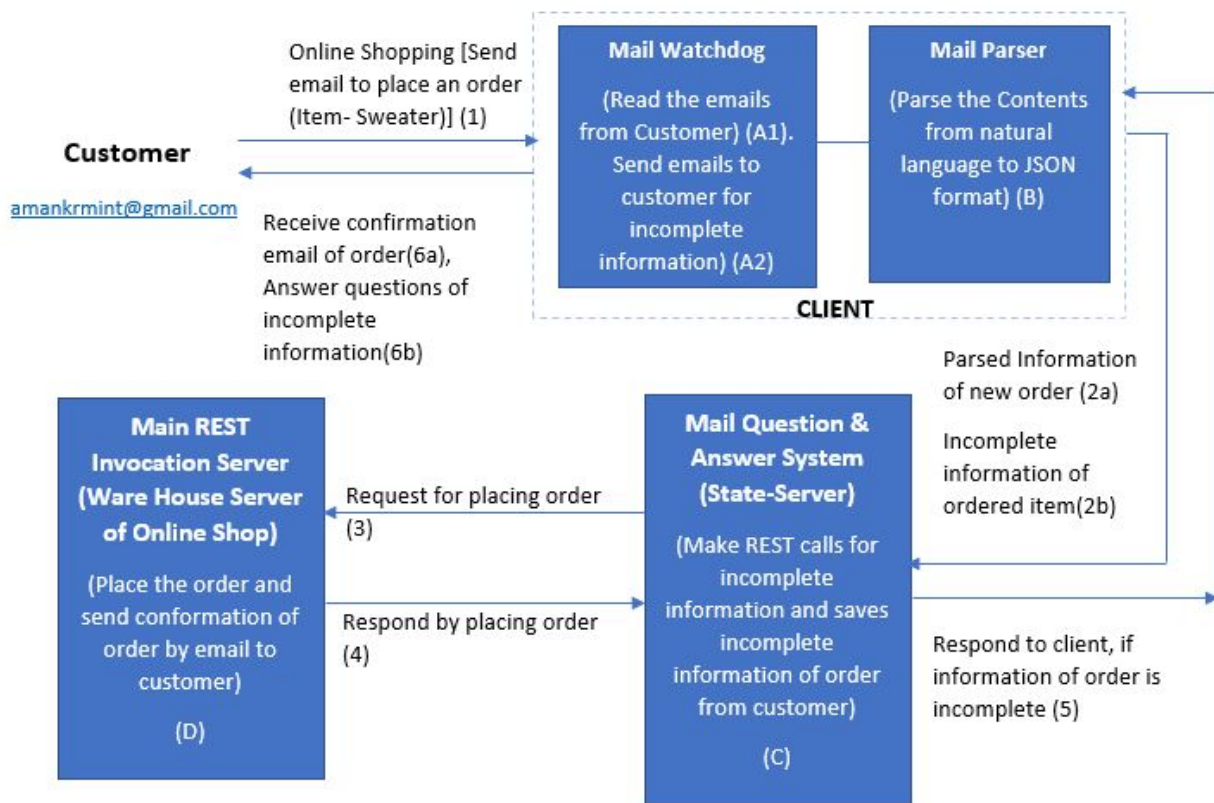


Figure 2 Order placement process of the E2R system at online shop

The functioning of this system is demonstrated by assuming a customer named Aman, who wants to purchase an item Sweater from Jaya's online shop (client) that is an online garment shop using the E2R User-Request Translation system (Figure 2). When customer Aman having email address, for example, amankrmint1@gmail.com (1 in Figure 2) makes a request for an order of item i.e., Sweater, email to online shop (Jaya's shop) i.e., jayamintk@gmail.com which has information of item requested by Aman in natural language (Figure 2.1).

Consider a case that customer [(1) in Figure 2] provides all information in email, required to place the order at online shop in email as shown in Figure 2.1. In this example of well-formed email, customer provides all details to online shop for an

order of item, such as color, design number, quantity and category of item and her shipping address.

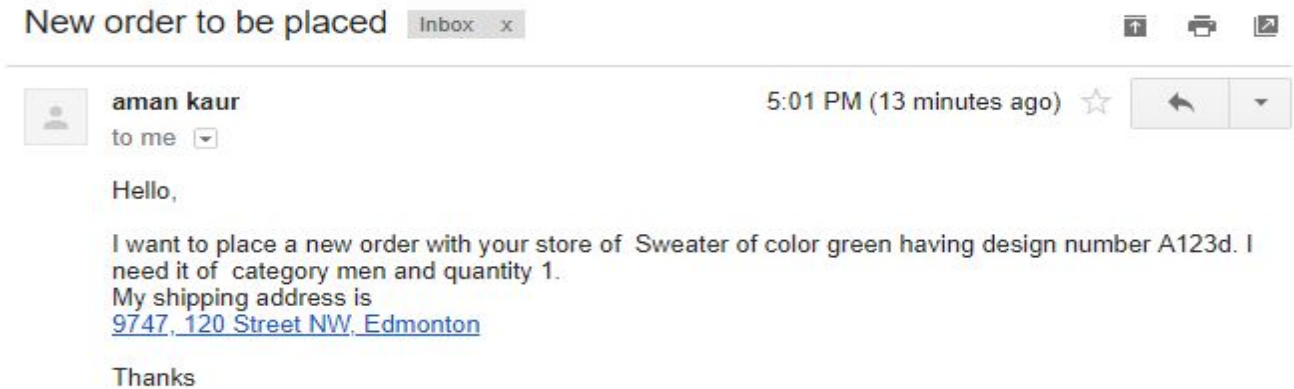


Figure 2.1 Well-formed email from customer to online shop (Jaya’s Shop)

Figure 2.1 is well-formed email from customer to online shop for placing an order of it Sweater, same like the template of ‘New order’ placed by customer of shopify.com (Figure 2.2) used to send new order notification to online shop when a customer places an order but in text form.

Hello jayamintk,

John Smith placed a new order with your store, Mar 12 02:57PM:

- 1x Aviator sunglasses (SKU: SKU2006-001) for \$89.99 each
- 1x Mid-century lounger (SKU: SKU2006-020) for \$154.99 each

[View order #9999](#)

Payment processing method:
visa, bogus

Delivery method:
Generic Shipping

Shipping address:
Steve Shipper
123 Shipping Street
Shippington, Kentucky 40003
United States
555-555-SHIP

Figure 2.2 ‘New order’ notification template of shopify.com when customer places an order

The Mail Watchdog [(A1) in Figure 2] of the client monitors the email inbox in a regular interval of time automatically, watches new emails 'UNSEEN' as shown in line 54 of code in client.py file in Figure 2.3.

```
52     print("Scanning new emails in inbox ")
53     # search for new unseen emails
54     result, data = mail.uid('search', None, 'UNSEEN')
```

Figure 2.3 Code to check unseen emails in inbox of the client in client.py file

The Mail Watchdog responds periodically, when a new email is received and gets notification of new email (new order) as shown in Figure 2.4 that shows the main function (defined in line 265) in Figure 2.4, regularly checks unseen emails after 60 seconds as program suspend execution of main function for 60 seconds demonstrated in line 277 of main function of the client.py script file (Figure 2.4).

```
264     # main function
265     def main():
266         mail, server = email_login()
267         while True:
268             status, data = search_inbox(mail)
269
270             if status:
271                 # If emails are found in inbox, Check all emails for orders and
272                 #answers of missing information of already placed order
273                 for email_uid in data[0].split():
274                     check_order_in_email(mail, email_uid, server)
275                 .
276             print("Email Checking Done.Checking again in 60 seconds")
277             time.sleep(60)
```

Figure 2.4 Code to check unseen emails in inbox periodically in client.py file

The Mail Watchdog of the client (Jaya's online shop) reads the contents of email that contain order information in plain text sent by customer (Figure 2.1) that is in natural human language from unseen emails in inbox.

```

107     # "New Order" template read and parse of shopify.com
108     def parse_new_order_template(server, email_message):
109         # print(email_message)
110         # a loop through all parts of email
111         for part in email_message.walk():
112             if part.get_content_type() == "text/plain": # only text of email
113                 body = part.get_payload(decode=True)
114                 content = body.decode('utf-8')

```

Figure 2.5 Email read function code at the client from customer in client.py file

The function defined at line 108 of code in client.py file of the client read the all contents of email (plain text) from customer as in Figure 2.5 and pass to the Mail Parser to extract useful information

The useful information of email is parsed by the Mail Parser of the client [(B) in Figure 2] and converted into JSON (JavaScript Object Notation) representation using regex expression method of parsing information. For example, in Figure 2.6, line 99 and 100 of code in client.py file of the client, parsing email using regex, looking for design number of Sweater, for example, Design Number is A123d of Sweater is found by searching of specific pattern (alphabetic, numeric, numeric, and numeric and alphabetic) in plain text of email received from customer. Similarly, client's Mail Parser looks for all other arguments (such as color, quantity and category) of item Sweater (ordered item) and parses them for placing an order.

```

99     designNumberMatches = re.findall('[a-zA-Z][0-9][0-9][0-9][a-zA-Z]', content)
100     order['Design Number'] = designNumberMatches[0] if designNumberMatches else None

```

Figure 2.6 Code of parsing of design number of ordered item Sweater using regex expression in client.py file

This parsed information [(2a) in Figure 2] from the client (online shop) is passed through the Mail Q and A System State-Server [(C) in Figure 2] which is a REST endpoint server using web framework Flask.

If parsed information has any incomplete information of order, the Mail Q and A system ask questions by making REST calls automatically to the client for clarification of missing arguments of order requested before finalizing the REST invocation to the Main Warehouse Server.

In this case, the Mail Q and A State-Server (Figure 3) stores the incomplete information of order and make the REST calls to the client [(5) in Figure 2] which will send email to customer [(A2) in Figure 2] as complete information is not taken from customer for a placed order. When customer responds with clarified information by email (6b in Figure 2) by answering the missing arguments of already placed order, contents of email are again read and parsed [(6b)→(A)→(B)→(2b) in Figure 2], and then passed through the Mail Q and A System (State-Server). At the same time, complete information is given by the customer through email, the Mail Q and A System make the REST call to the Main Warehouse Server working on the web framework Flask [(3) in Figure 2] to place the order which is actually responding to customer's request [(1) in Figure 2], places the order by responding to REST call with REST endpoint function defined at line 26 of Figure 2.7 of script of server.py file.

```
25 @app.route("/order", methods=['POST'])
26 def add_order():
27     data = request.json
28
29     print("Order received")
30     print(json.dumps(data, indent=4))
31
32     orders_db[data['Order Number']] = data
33
34     return jsonify({'Status': 'OK', 'Number of Orders': len(orders_db.keys())}), 200
35
```

Figure 2.7 Code of REST function at the Main Warehouse Server respond to request for placing an order in sever.py file

After that, Customer is notified by sending confirmation back to the customer by the client [(6b) in Figure 2] using same ‘Order confirmation’ template of e-commerce company shopify.com (Figure 2.8) but in text form.

jayamintk ORDER #9999

Thank you for your purchase!

Hi John, we're getting your order ready to be shipped. We will notify you when it has been sent.

[View your order](#) or [Visit our store](#)

Order summary

Aviator sunglasses × 1	\$89.99
Mid-century lounge × 1	\$159.99 \$154.99
Subtotal	\$244.98
Shipping	\$10.00
Total	\$254.98 CAD

Customer information

Shipping address	Billing address
Steve Shipper	Bob Biller
Shipping Company	My Company
123 Shipping Street	123 Billing Street
Shippington KY 40003	Billtown KY K2P0B0
United States	United States
Shipping method	Payment method
Generic Shipping	Ending in 1234 — \$254.98

If you have any questions, reply to this email or contact us at jayamintk@gmail.com

Figure 2.8 ‘Order Confirmation’ notification to customer template of shopify.com


The client (Jaya's online shop) send confirmation email in natural language as demonstrated in function 'order_confirmation_email' in line 160 of code in client.py file of the client as shown in Figure 2.9. This function sends order notification to only customer [(6a) in Figure 2], when the Main Warehouse Server responds with successful placement of order containing order confirmation notification with order details to customer as in 'message =' of line 161,162 to 169 of code in client.py file in text form.

```
160 def order_confirmation_email(server, target_email, order_number, data):
161     message = 'ORDER #{}'.format(order_number)
162     message += '\n\n Thank you for your purchase! \n\n Hi {}, We're getting your order
163     ready to be shipped.We will notify you when it has been sent.'.format(data['Customer Name'])
164     order = data['Order']
165     message += '\n\n Order summary \n Item Name : %s \n Color : %s \n Design Number : %s
166     \n Category : %s \n Quantity : %s \n Shipping Address : %s \n' % tuple(
167         [order['Name'], order['Color'], order['Design Number'], order['Category'], order['Quantity'],
168         order['Shipping Address']])
169     message += '\n\n If you have any questions,reply to this email or contact us at jayamintk@gmail.com '
170     email_msg = MIMEText("{}".format(message))
171     email_msg['Subject'] = "Order #{} confirmed".format(order_number)
172     email_msg['From'] = "jayamintk@gmail.com"
173     email_msg['To'] = target_email
174     server.sendmail("jayamintk@gmail.com", target_email, email_msg.as_string())
```

Figure 2.9 Order Confirmation email to customer code in client.py file

The E2R system sends confirmation email back to customer of placed order via email in natural language (text form) from the client (Jaya's online shop) as shown in Figure 2.10 containing order number, notification of confirmed order with details of ordered item Sweater, order status and shipping address to customer (Aman).

Order #7 confirmed Inbox x

 **jayamintk@gmail.com**
to me

ORDER #7

Thank you for your purchase!

Hi aman kaur, We're getting your order ready to be shipped. We will notify you when it has been sent.

Order summary
Item Name : sweater
Color : green
Design Number : A123d
Category : men
Quantity : 1.
Shipping Address : 9747, 120 Street NW, Edmonton
If you have any questions, reply to this email or contact us at jayamintk@gmail.com

Figure 2.10 ‘Confirmation of Order Email’ notification to Customer Aman from the client (Jaya’s online shop)

Example of Mail Q and A System

Considering the same example of customer Aman, who wants to order an item Sweater, from Jaya’s online shop. In the general proper scenario (Figure 3), customer [(A) in Figure 1] gives all information for item, i.e., all arguments required for placing an order of item. In this scenario, when customer makes a request by sending an email to Jaya’s online shop, email read by the Mail Watchdog and contents of email are parsed by the Mail Parser and request passed to the Mail Q and A State-Server which checks that all information is complete and passes the request to the Main Warehouse REST Server for placing an order requested from customer by using REST Invocation. The Main REST Server places the order [follows path (A)→(1)→(B1)→(C)→(D)→(E) in Figure 3] and send confirmation email notification back to customer [(iii) in Figure 3] for placed order at the same email address of customer used for placing the order.

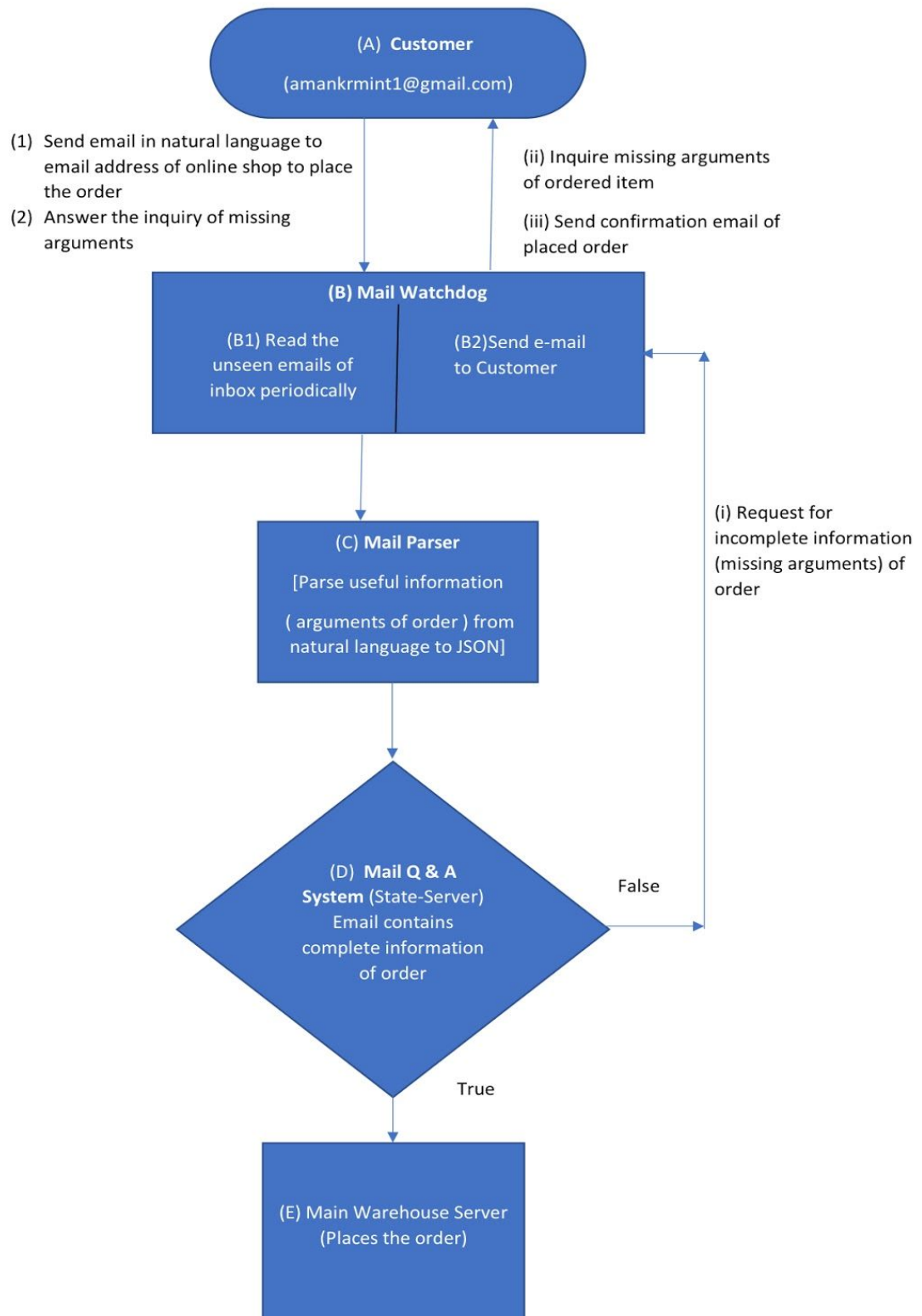


Figure 3 Flow chart of order placement process for the Mail Q and A system

In other scenario, when customer makes a request for an item, while placing the order, gives incomplete information of order i.e., some arguments of item are missed, for example, in Figure 3.1, quantity and color of item Sweater was not provided (incomplete information) by customer while placing the order and email was sent to the online shop.



Figure 3.1 Incomplete order information email from customer to client (Jaya's online Shop)

In this case, when customer's email contents parsed by the client (Jaya's online shop), makes a REST call to the Mail Q and A System which is State-Server before placing the order. The Mail Q and A System responds back to the client for missing arguments that are checked at the Mail Q and A System [(D) in Figure 3] by invoking 'get_missing()' REST endpoint function defined at line 30 of code in qna.py file shown in Figure 3.2 which returns missing arguments of current order.

```
29 @app.route('/qna', methods=['GET'])
30 def get_missing():
31     data = request.json
32     print("request: {}".format(data))
33     orderNumber = int(data['Order Number'])
34     if orderNumber in orders_db.keys():
35         missing = list(set(['Name', 'Color', 'Design Number', 'Category',
36                             'Quantity', 'Shipping Address']).difference(list(orders_db[orderNumber].keys())))
37         print("Order details found: {}".format(orders_db[orderNumber]))
38         return jsonify({'Order': orders_db[orderNumber], 'Missing': missing}), 200
39     else:
40         return jsonify({'STATUS': "Failed"}), 404
```

Figure 3.2 REST endpoint function to check all arguments are in parsed information in the Mail Q and A system(State-Server) in qna.py script file

An email (Question) to customer (Aman) for clarification of missing arguments is sent by the client (Jaya's online shop) as shown in Figure 3.3, in response of False condition of the Mail Q and A System [(D)→(i)→(B2)→(ii)→(A) in Figure 3], i.e., when complete order details are not found.

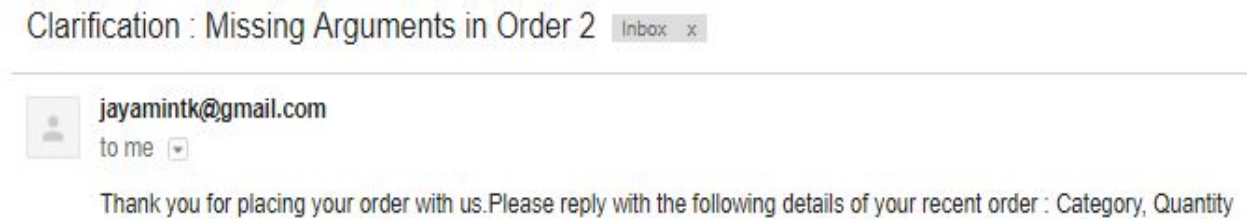


Figure 3.3 Email sent by the client to customer to inquire missing information of placed order

When customer answer the incomplete information of placed order [(2) in Figure 3]. A request from the client to the State-Server is made for missing arguments case following the same process of read the email and parse the arguments, and the client gives the missing arguments from customer to State-Server [follows path (A)→(2)→(B1)→(C)→(D) in Figure 3] which append these missing arguments in existing incomplete order as shown in Figure 3.4 in script of qna.py file where a REST function 'append_to_order' is defined at line 44 of qna.py file, appending the information given by the client into current incomplete order.

```
43 @app.route('/qna/append', methods=['POST'])
44 def append_to_order():
45     data = json.loads(request.json)
46     print(data)
47     print(type(data))
48     orderNumber = int(data['Order Number'])
49     print("Recieved append request: {}".format(data))
50
51     if data['Order']:
52         for key, value in data['Order'].items():
53             orders_db[orderNumber][key] = value
54
55     missing = list(set(['Name', 'Color', 'Design Number', 'Category',
56 'Quantity', 'Shipping Address']).difference(list(orders_db[orderNumber].keys())))
57
58     return jsonify({'Order': orders_db[orderNumber], 'Missing': missing}), 200
59
```

Figure 3.4 Code of Mail Q and A System (State- Server) to respond the request from the client to State-Server for appending missing information in qna.py script file

Now further two cases are possible:

1. Customer responds with answer of all incomplete information [(A)→(2) in Figure 3], i.e., all missing arguments, quantity of items and color of item which is inquired by the Mail Q and A System. In this case, when the Mail Q and A System (State-Server) will get the response(Answer) back from the customer, through the parsed arguments passed by the client to State-Server, combines the information saved in it already for order and missing information given by the client now and will make a REST call to Warehouse REST Server [follows path (A)→(2)→(B1)→(C)→(D)→(E) in Figure 3]. In the meantime, order will be placed at Warehouse REST Server and a confirmation email with ordered information will be sent back on the email address of customer in natural language which is in user readable format[(iii) in Figure 3].

2. Customer responds with incomplete information [(A)→(2) in Figure 3], i.e., some missing arguments, only quantity of item, but did not provides information of color of item which was also inquired by the Mail Q and System. For this scenario, the Mail Q and A System saves information of existing order and information answered by customer (Figure 3.4) and makes a REST call again to the client to send an email to customer for incomplete information (color of item) follows again the same loop as discussed above [again (D)→False→(i)→(B2)→(ii)→(A) in Figure 3]. As soon as customer replies with all left information required to complete the process of order placing to the client, order is placed at Warehouse Server and confirmation email with ordered information will be sent back on the email address of customer as discussed in Case 1[(A)→(2)→(B1)→(C)→(D)→(E) in Figure 3]; otherwise, the Mail Q and A System again requests to the client for missing information and the client again sends email (Question) to customer for incomplete information and waits for reply from customer to complete the order information and place the order.

How is Capstone Project different from Wenting Zhang?

My working component of the E2R System is completely different from Wenting Zhang's, and also I haven't seen her code during the project completion. Hence, my code is completely separate from her code.

The E2R system architecture is main idea of this MINT Capstone Project on which multiple students are working on different or combination of components of overall framework. My project is focused only on the Mail Question and Answer (Q and A) System component based on particular example that works to clarify any incomplete information from customer by the client for placing an order. My classmate Wenting Zhang worked on the Mail Watchdog component which responds to new incoming emails and the Mail Parser component extract the Email and translate human language to useful data (text to JSON), that is a totally different component from my chosen Mail Question and Answer System component which is the State-Server of the E2R System.

Conclusion

This project uses the E2R system which is based on Representational State Transfer (REST) architecture. In this system, a Mail Watchdog monitors mailbox to check and respond new mails, based on the email contents, it calls the Mail Parser which translate human language to useful data (i.e., plain text to JSON) parse the contents of email to JSON file. If the Mail Parser contents does not contain all information or has questions about the specific REST invocation, the Mail Q and A system (State-Server) make a REST call to the client for incomplete information of order which inquire missing information from customer through email. This request and response session is in JSON file format between the client and endpoint servers. After processing of request, Main Server of the E2R System responds in computer language. The E2R system sends confirmation back to customer via email in natural language.

References

[1] <https://en.wikipedia.org/wiki/Shopify>

[2] <https://www.juliedesk.com>

[3] <http://www.drdoobbs.com/web-development/restful-web-services-a-tutorial/240169069>

[4] http://www.restapitutorial.com/media/RESTful_Best_Practices-v1_1.pdf

[5] <https://mailparser.io/>

[6] <http://docs.python-requests.org/en/master/user/quickstart/>

[7] <https://apps.shopify.com/simp-questions-and-answers>

[8] <https://jayamintk.myshopify.com/admin/settings/notifications>

[9] Flask: Building Python Web Services By: Gareth Dwyer; Shalabh Aggarwal; Jack Stouffer Publisher: Packt Publishing

[10] <https://stackoverflow.com>

[11] <http://www.restapitutorial.com/lessons/whatisrest.html>

[12] https://www.tutorialspoint.com/flask/flask_file_uploading.htm

[13] <http://flask.pocoo.org/>