Mint Capstone Project Final Report

Email-to-REST User-Request Translation System

Instructor : Paul Lu
Student : Wenting Zhang

Abstract

In recent years, online shopping has become an important shopping method in people’s life. Some online shopping platform use emails to inform shop owners of order details. Therefore, it is convenient to have a special system to help process order emails automatically. We implemented a prototype Email-to-REST User-Request Translation System (referred to as the E2R system below) that receives email orders and automatically adds the order into a company database. In the prototype, a Shopify-like Web server (implemented using Django, Figure 2.1.1 (A)) can send an email representing a new customer order. Five email order templates are supported. The the E2R system Figure 2.1.1 (B)) reads the email (from Gmail.com) and adds the order to a company database (implemented using Flask, Figure 2.1.1 (C)).

1. Background

1.1 Shopify Introduction

In recent years, eCommerce has become an important part of people's lives and an important business transaction method. With the trend of eCommerce, there comes some new-concept online shopping platforms. These platforms are specially designed for individuals and small businesses. Shopify.com is such a new online shopping platform. Individuals or small shops on Shopify can easily own their exclusive online shops, with special URLs and DIY home pages. After customers click into these online shops and place orders (Figure 1.1.1 ①), a transaction summary will be sent to both the customer and the shop owner via emails (Figure 1.1.1 ②).

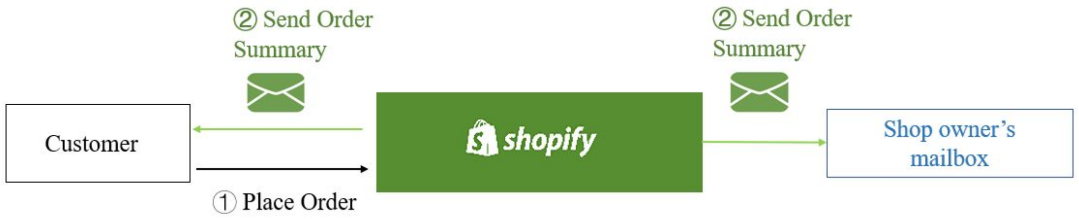
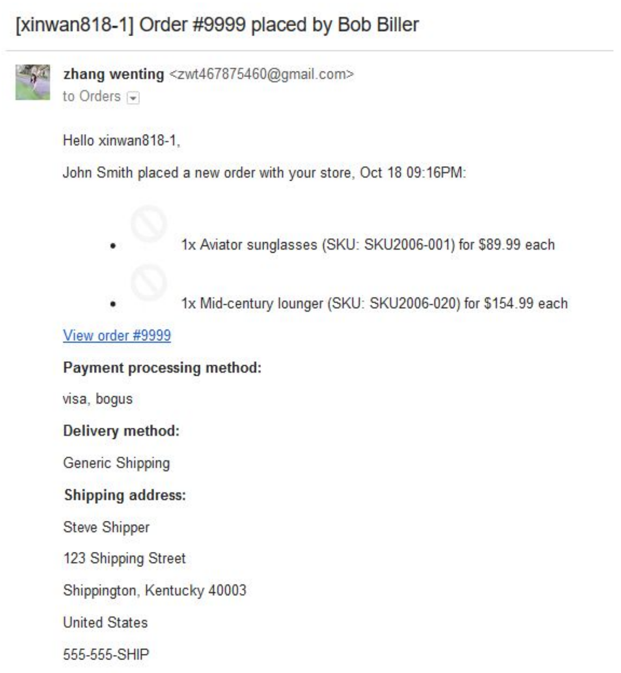


Figure 1.1.1 One customer places an order on Shopify and Shopify send order summary to the customer and shop owner (Black for customer's actions, green for Shopify's actions)

As for the email from Shopify, Shopify has special templates of emails for different uses. After a new order has been placed, order details will be rendered into some special email templates and sent to users. If the shop owner does not make any changes to Shopify's default email template, the template for the shop owner will be like Figure 1.1.2(A). Besides, shop owners can design their own email templates on users' control panels. If a new template is updated by one user, order details will be rendered into the user's new email template (Figure 1.1.2(B)) and sent to his email.


[xinwan818-1] Order #9999 placed by Bob Biller

zhang wenting <zwt467875460@gmail.com>
to Orders

Hello xinwan818-1,

John Smith placed a new order with your store, Oct 18 09:16PM:

- 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:

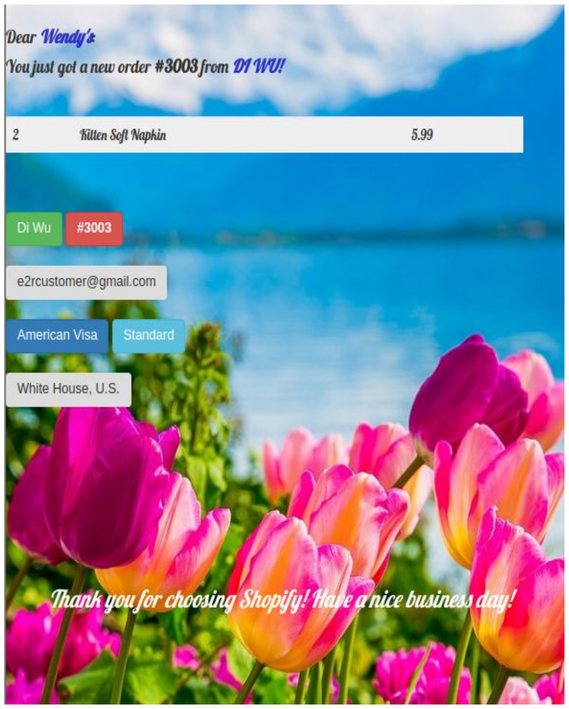
Steve Shipper

123 Shipping Street

Shippington, Kentucky 40003

United States

555-555-SHIP


Dear Wendy's

You just got a new order #3003 from DI WU!

2	Kitten Soft Napkin	5.99

Di Wu #3003

e2rcustomer@gmail.com

American Visa Standard

White House, U.S.

Thank you for choosing Shopify! Have a nice business day!

(A) Shopify's default Email template for order summary **(B)** One user's DIY template

Figure 1.1.2 Email template samples

1.2 Motivating Problem

Although the business model of Shopify is great, it calls for a lot of human actions which can be error-prone and usually cause high latency. With the rapid development of online shopping, it is common that buyers from all over the world may place thousands of orders in an online shop in one day. In Shopify's example, if one shop on Shopify gets 1000 orders one day, the small business owner needs to deal with 1000 emails and checks every single item in his warehouse manually. In this way, a system to help people "read" orders from emails, record new orders and user information into a database, and trigger the shipping will be good to people's life.

As in Figure 1.2.1, the E2R is designed to process orders (Figure 1.2.1 (1) "Process Orders") and send process reports (Figure 1.2.1(2) "Send Reports") to shop owners by polling emails from shop owner's mailboxes (Figure 1.2.1a), translating emails (Figure 1.2.1b) to REST calls (Figure 1.2.1c).

The "Translation" idea is similar to Mailparser.io [1] and juliedesk.com [2], which automatically parses emails with a pre-designed, semi-structured layout and then takes an appropriate action (e.g., extracting user-specified data for Mailparser.io and scheduling meetings for juliedesk.com) on a back-end server. Different from Mailparser.io and juliedesk.com, the E2R is exclusively designed for online shopping (especially for Shopify users). So the features focus more on online shopping needs. Plus, the template process algorithm of the E2R is designed to be flexible. The raw email can be Text and HTML. To gain higher system performance, email can also be JSON like format (A sample can be seen in Figure 1.2.2).

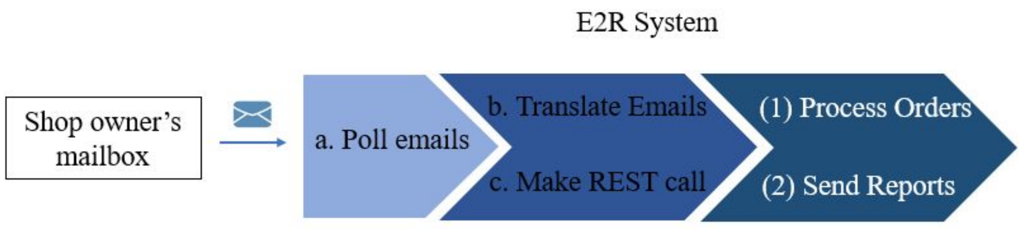


Figure 1.2.1 the E2R process Email for users and send report

```
{
    "shops" : [
        {
        "Shop" : "Wendy's",
        "Date" : "2017-09-15 Fri",
        "Command" : "Place Order";
        }
    ],

    "items": [
        {
        "Item" : "Pop tart jumbo",
        "Size" : None,
        "Color" : None,
        "Num" : "3"
        },
        {
        "Item" : "broken jeans",
        "Size" : None,
        "Color" : "Blue",
        "Num" : "1"
        }
    ],

    "customers" : [
        "Name" : "John Smith",
        "Street No." : "1221 Jasper Avenue",
        "City" : "Edmonton",
        "State" : "Alberta"
        ]
  }
```

Figure 1.2.2 Order JSON file sample

2. Introduction

The working environment of the E2R system contains three main parts: the Shopify website (Figure 2.1.1(A)), the the E2R system (Figure 2.1.1(B)) and a third party server (Figure 2.1.1(C)).

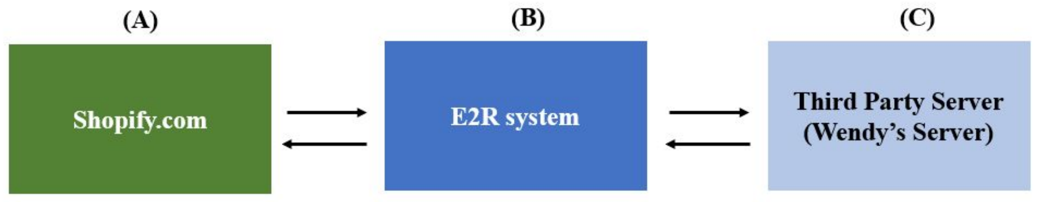
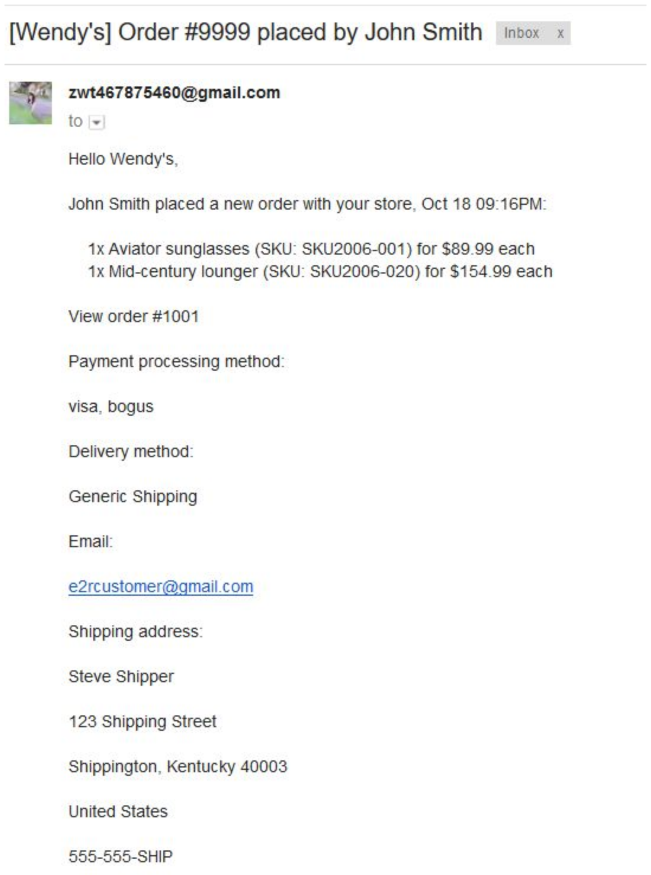


Figure 2.1.1 System architecture

To make the description clear, let us put the E2R model into a simple scenario: John Smith wants to buy some products in Wendy's shop. Wendy's shop is one online shop on Shopify.com, which has its own warehouse server and refers to as the third party server ((Figure 2.1.1(C)) in this example.

First of all, Shopify sends an order summary to the shop owner's mailbox (2.1.3(1)). In John Smith's case, his order details and personal information (e.g. shipping address) will be sent to Wendy's mailbox. A sample Email can be seen in Figure 2.1.2.


[Wendy's] Order #9999 placed by John Smith Inbox x

zwt467875460@gmail.com
to

Hello Wendy's,

John Smith placed a new order with your store, Oct 18 09:16PM:

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

View order #1001

Payment processing method:

visa, bogus

Delivery method:

Generic Shipping

Email:

e2rcustomer@gmail.com

Shipping address:

Steve Shipper

123 Shipping Street

Shippington, Kentucky 40003

United States

555-555-SHIP

Figure 2.1.2 Order summary of John Smith sent to Wendy's mailbox

Then, the the E2R system polls emails from the shop owner's mailbox. Usually, the the E2R system is set to monitor the shop owner's' mailbox (Wendy's mailbox: xinwantest@gmail.com in this case) so once the E2R system finds there are unread mails, it will start to poll emails from W(Figure 2.1.3 (2)).

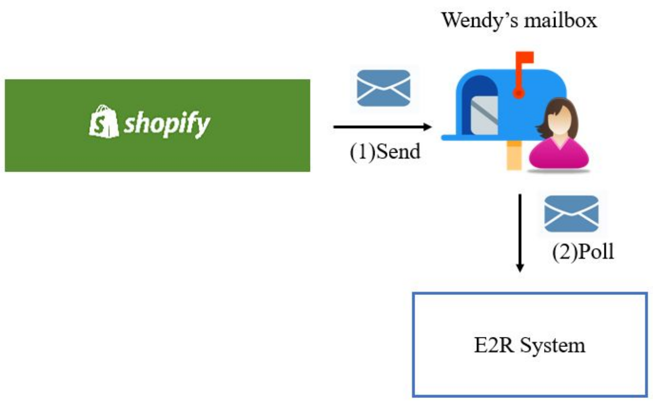


Figure 2.1.3 Process of the E2R getting raw emails

Afterwards, as shown in Figure 2.1.4 E1(E1 for Phase 1 of E2R), the E2R system will translate emails (MIME file in natural language) into a JSON file, and use the JSON file to make a REST call (E2 of Figure 2.1.4) to Wendy's server, which will trigger Wendy's server to deal with the order in Wendy's system (Figure 2.1.4 W1, W1 stands for Phase 1 of Wendy's server). When Wendy's server has processed the order and put it into the database, it will send a response back to the E2R system (Figure 2.1.4 W2). Details about the two phases will be discussed in Section 3.3, and the technical process in Wendy's server will be discussed in Section 3.2.

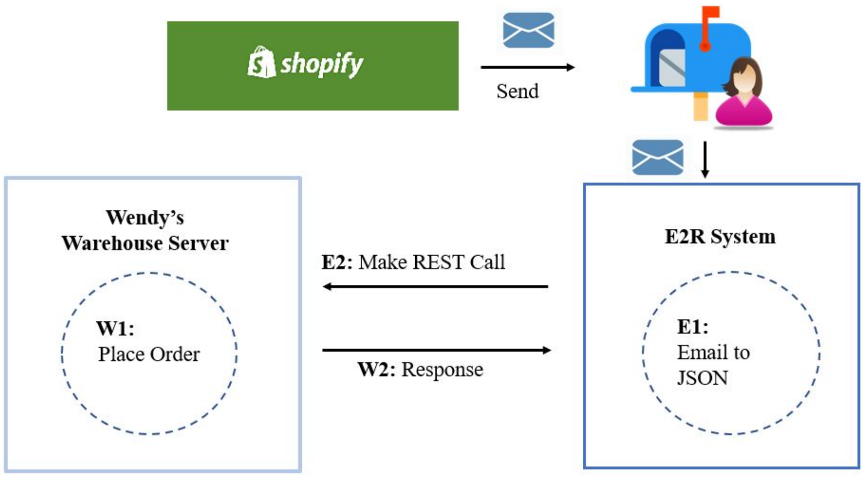


Figure 2.1.4 Interaction with Wendy's server

After receiving the response (the JSON file) from the third party server (Wendy's server), the E2R will translate the JSON file into MIME format in natural language (E3 of Figure 2.1.5). Then the E2R sends a confirmation email to John Smith (E4(a) of Figure 2.1.5) and a report to Wendy (E4(b) of Figure 2.1.5). The email samples are in Figure 2.1.6.

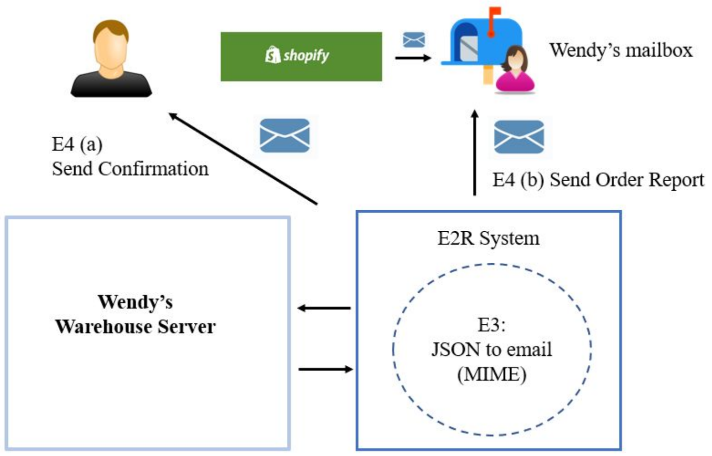
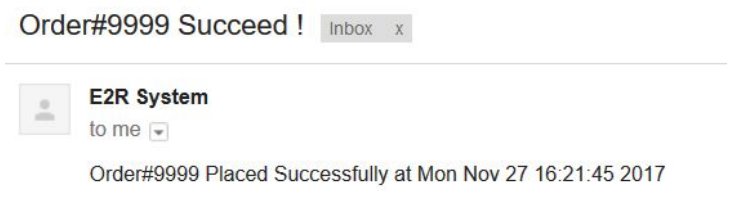


Figure 2.1.5 the E2R translate Wendy's server's response into email and interact with users

Sometimes, customers may change their minds after the order has been placed. Therefore, when the customer (John Smith) receives the confirmation email (Figure 2.1.6(B)), the customer can still make changes to his order or cancel it. If the customer clicks "change" (lower left corner in Figure 2.1.6(B)), he will be redirected to Wendy's server to fill in an update form (Figure 2.1.7). Afterwards, the updated information will be sent to Wendy. Besides, if the customer cancels the order, the cancellation request will also be sent to Wendy.

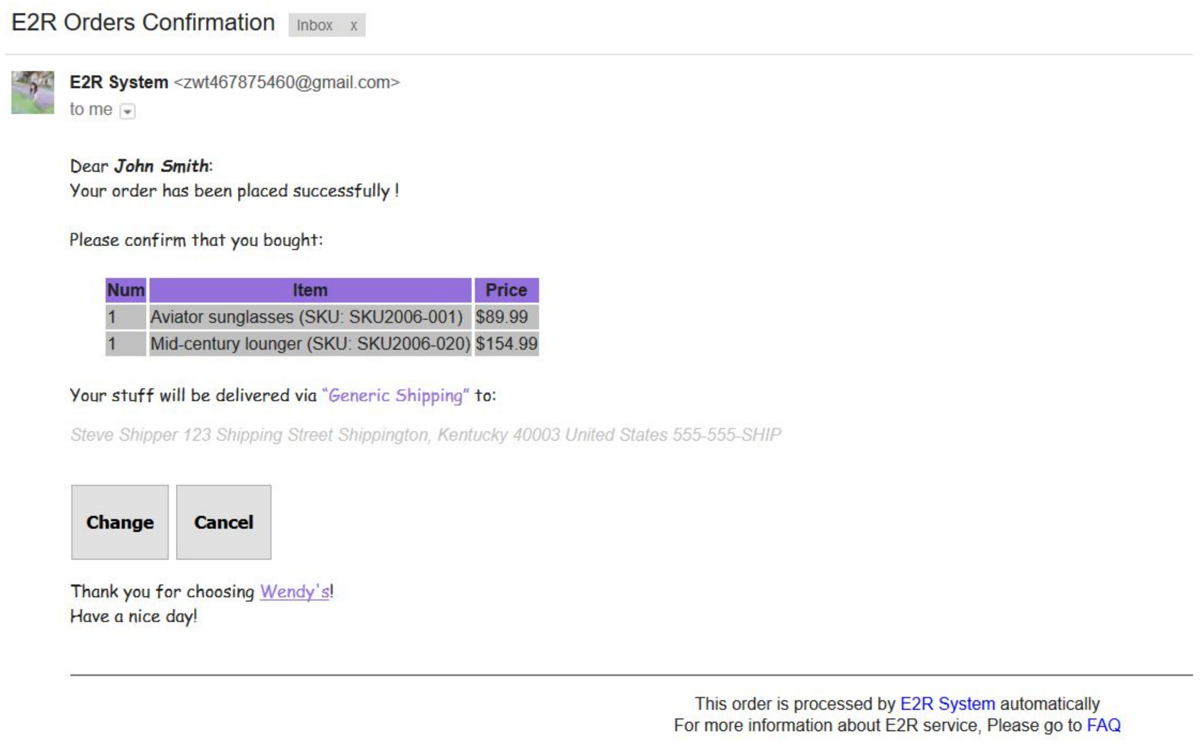


Order#9999 Succeed !

E2R System
to me

Order#9999 Placed Successfully at Mon Nov 27 16:21:45 2017

(A) Order report in Wendy's mailbox



E2R Orders Confirmation

E2R System <zwt467875460@gmail.com>
to me

Dear ***John Smith***:
Your order has been placed successfully !

Please confirm that you bought:

Num	Item	Price
1	Aviator sunglasses (SKU: SKU2006-001)	$89.99
1	Mid-century lounger (SKU: SKU2006-020)	$154.99

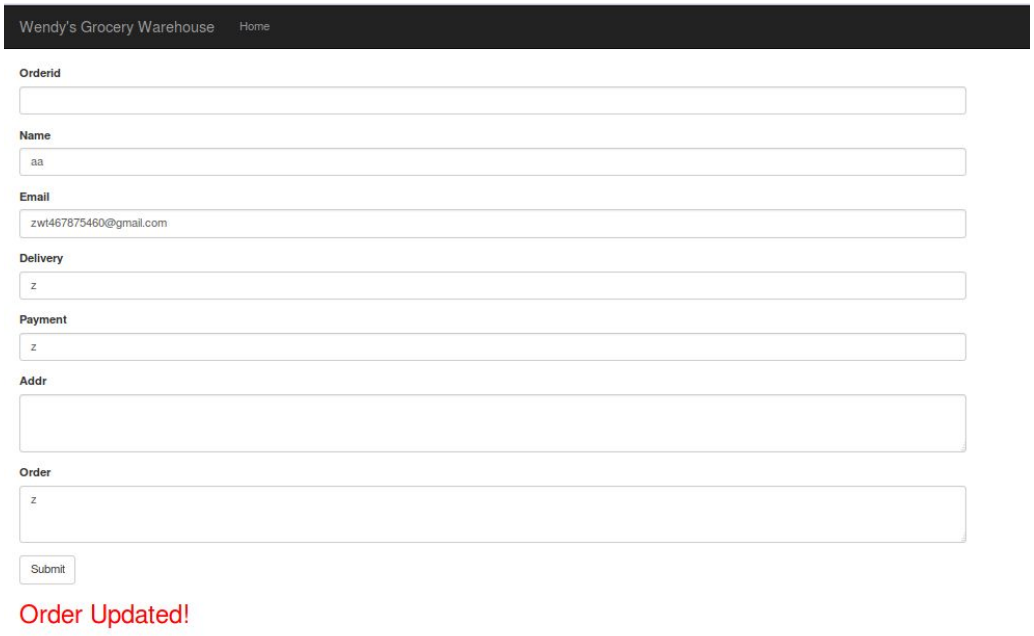
Your stuff will be delivered via "Generic Shipping" to:

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

Thank you for choosing Wendy's!
Have a nice day!

This order is processed by E2R System automatically
For more information about E2R service, Please go to FAQ

(B) Confirmation email sent to John Smith
Figure 2.1.6 Emails from the E2R system


Wendy's Grocery Warehouse Home

Orderid

Name
aa

Email
zwt467875460@gmail.com

Delivery
z

Payment
z

Addr

Order
z

Submit

Order Updated!

Figure 2.1.7 Update Form in Wendy's server

Instead of letting a customer make changes directly to orders which are already in database and maybe under process, all update or cancellation requests will be sent to Wendy's mailbox to be processed manually. A flowchart of the process is shown in Figure 2.1.8 and details will be discussed in Section 3.2.

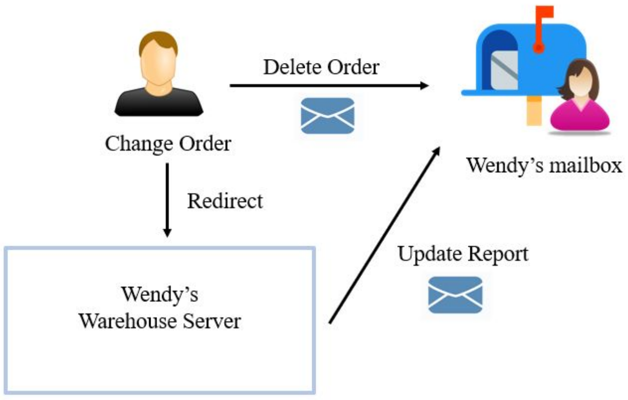


Figure 2.1.8 Customer operate orders with confirmation email

3. System Design

To simulate the E2R working environment and test the E2R system, the MINT Capstone Project implements a mock Shopify website (Figure 2.1.1 (A)), an E2R system (Figure 2.1.1 (B)) and a mock Wendy's server (Figure 2.1.1 (C)). This Section will introduce the design and main functions of the three systems.

3.1 Shopify

Shopify is implemented by building a simulated website based on the Django framework [6] (Part 1 in Figure 3.1.1) to send an order summary email in two modes. For testing, a file named "sender.py" and a module "mailProcessor" are implemented to send 5 kinds of email templates in 5 modes (Part 2 in Figure 3.1.1). This Section will only focus on the simulated website and the test program will be discussed in Section 5.2.

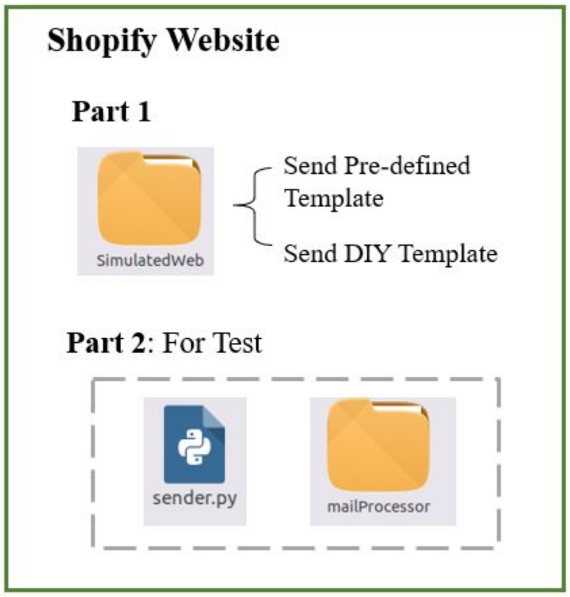


Figure 3.1.1 Files in Shopify

When the mock Shopify website has been launched, user can choose 2 modes to send 3 kinds of order summary emails to Wendy (Table 3.1.1). The sending feature is designed to simulate the Shopify sends order summary emails to shop owners. Besides, because users are allowed to use both the default and DIY email templates, the simulated Shopify also has two sending modes to do the same thing.

Mode	URL	Email Type
DIY Template	ShopifyURL.com/sender	/

Pre-defined Template	ShopifyURL.com/template/1	Text
	ShopifyURL.com/template/2	JSON
	ShopifyURL.com/template/3	HTML

Table 3.1.1 Two modes to send emails in Shopify and URL

Generally, commercial emails can be divided into 2 type: Text and HTML. Besides, because of the DIY template property of Shopify, it is convenient for the E2R system to translate "JSON" emails. Therefore, the simulated Shopify is designed to send the 3 kinds of emails: Text, HTML, JSON. Figure 3.1.2 shows DIY sending mode interface and Figure 3.1.3 shows the HTML Email sending interface.

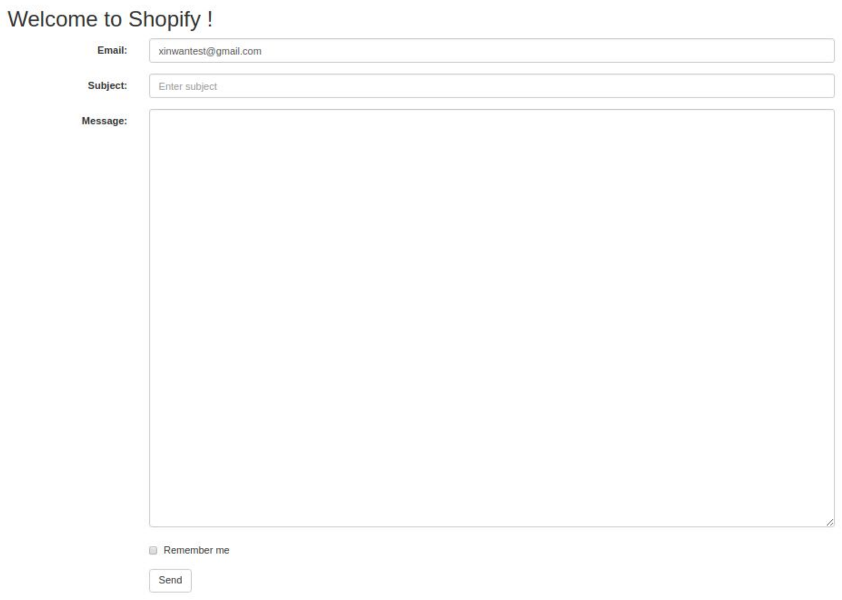
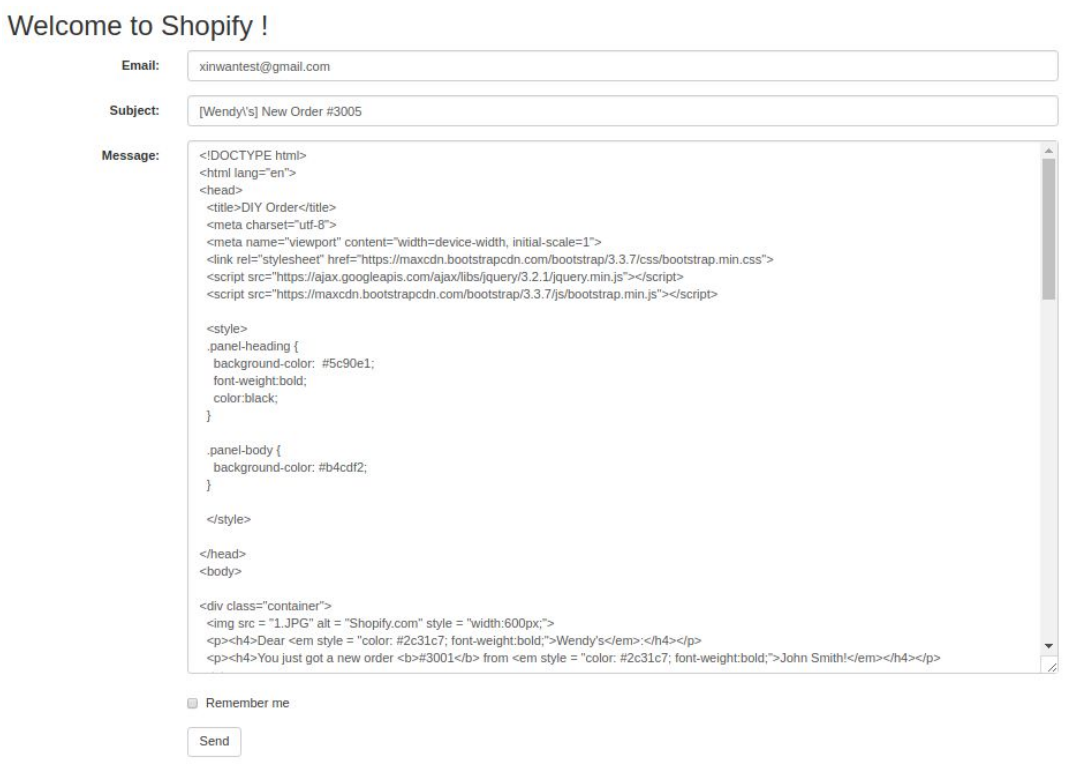


Figure 3.1.2 Shopify sends DIY Email template



Welcome to Shopify !

Email: xinwantest@gmail.com

Subject: [Wendy\'s] New Order #3005

Message:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>DIY Order</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

  <style>
  .panel-heading {
    background-color:  #5c90e1;
    font-weight:bold;
    color:black;
  }

  .panel-body {
    background-color: #b4cdf2;
  }

  </style>

</head>
<body>

<div class="container">
  <img src = "1.JPG" alt = "Shopify.com" style = "width:600px;">
  <p><h4>Dear <em style = "color: #2c31c7; font-weight:bold;">Wendy's</em>:</h4></p>
  <p><h4>You just got a new order <b>#3001</b> from <em style = "color: #2c31c7; font-weight:bold;">John Smith!</em></h4></p>
```

Figure 3.1.3 Shopify Sends Pre-designed HTML Email template

3.2 Wendy's Warehouse Server

Wendy's Warehouse Server is built based on Flask framework [4]. There are 4 main methods to deal with REST calls and two interfaces to deal with order update and cancellation requests. The four REST related methods (GET, POST, PUT, DELETE) are built on route "WendyURL.com/todo/api/orders" [5], functioning as API to external servers and controlling database inside. The "update" interface is built on route "WendyURL.com/change" and The "cancel" interface is built on route "WendyURL.com/cancel".

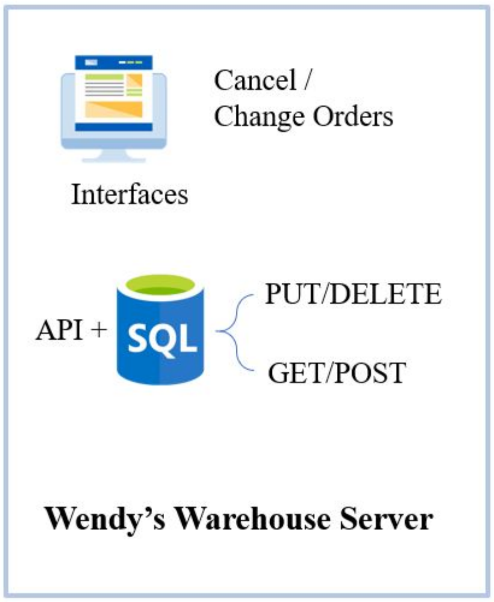
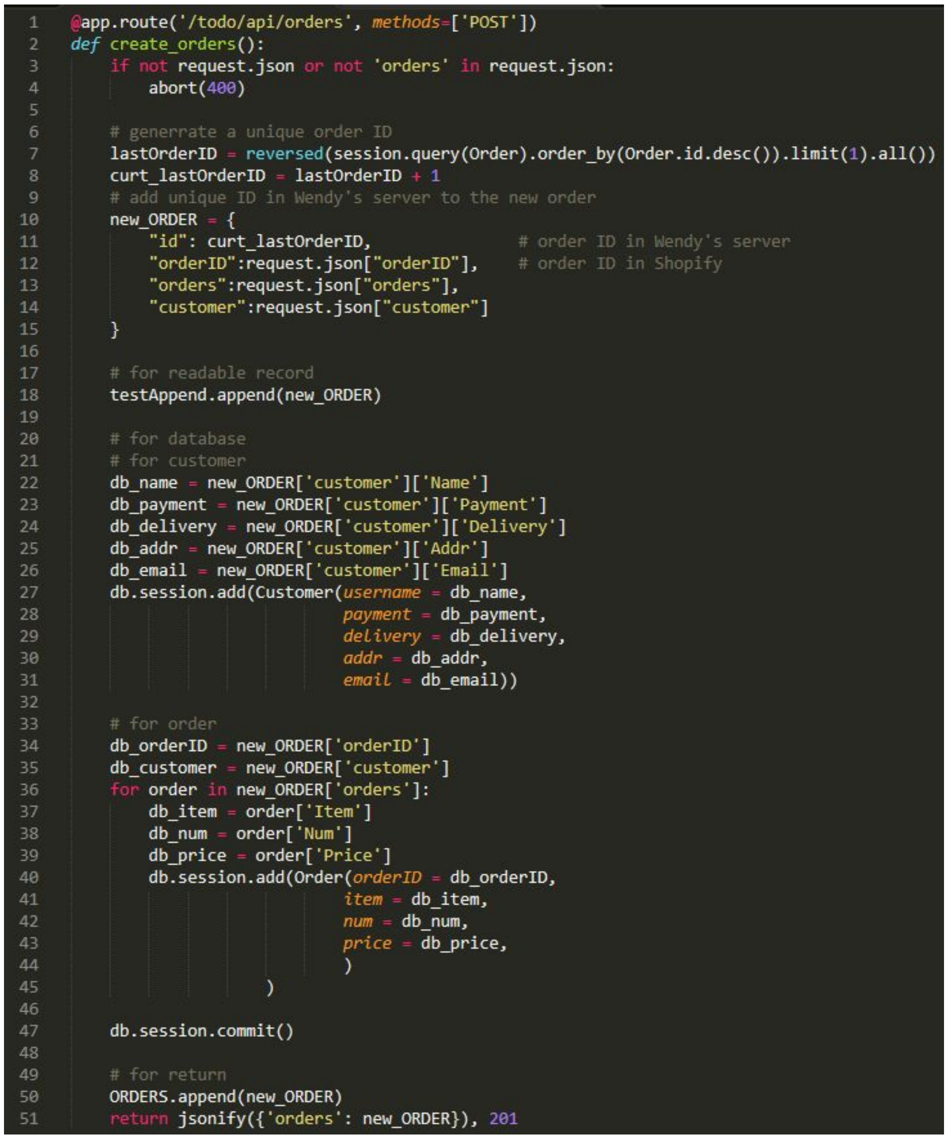


Figure 3.2.1 Main features of Wendy's server

Talking about the four REST call related features, if a remote client makes a POST call, Wendy's server will create a new order according to the information in this call. Figure 3.2.2 shows the function of POST method in Wendy's server, and Figure 3.2.3 shows database structure. In John Smith's example, when the E2R system posts an order request (a JSON file contains shipping information and order details) to Wendy's server by REST call (*method = POST*), a function named "create_orders()" (Figure 3.2.2) will be triggered to process the order.



```
1   @app.route('/todo/api/orders', methods=['POST'])
2   def create_orders():
3       if not request.json or not 'orders' in request.json:
4           abort(400)
5
6       # generrate a unique order ID
7       lastOrderID = reversed(session.query(Order).order_by(Order.id.desc()).limit(1).all())
8       curt_lastOrderID = lastOrderID + 1
9       # add unique ID in Wendy's server to the new order
10      new_ORDER = {
11          "id": curt_lastOrderID,                 # order ID in Wendy's server
12          "orderID":request.json["orderID"],      # order ID in Shopify
13          "orders":request.json["orders"],
14          "customer":request.json["customer"]
15      }
16
17      # for readable record
18      testAppend.append(new_ORDER)
19
20      # for database
21      # for customer
22      db_name = new_ORDER['customer']['Name']
23      db_payment = new_ORDER['customer']['Payment']
24      db_delivery = new_ORDER['customer']['Delivery']
25      db_addr = new_ORDER['customer']['Addr']
26      db_email = new_ORDER['customer']['Email']
27      db.session.add(Customer(username = db_name,
28                              payment = db_payment,
29                              delivery = db_delivery,
30                              addr = db_addr,
31                              email = db_email))
32
33      # for order
34      db_orderID = new_ORDER['orderID']
35      db_customer = new_ORDER['customer']
36      for order in new_ORDER['orders']:
37          db_item = order['Item']
38          db_num = order['Num']
39          db_price = order['Price']
40          db.session.add(Order(orderID = db_orderID,
41                              item = db_item,
42                              num = db_num,
43                              price = db_price,
44                              )
45                      )
46
47      db.session.commit()
48
49      # for return
50      ORDERS.append(new_ORDER)
51      return jsonify({'orders': new_ORDER}), 201
```

Figure 3.2.2 A function in Wendy's server which deals with POST request and sends back response (201: Created / 400: Bad Request)

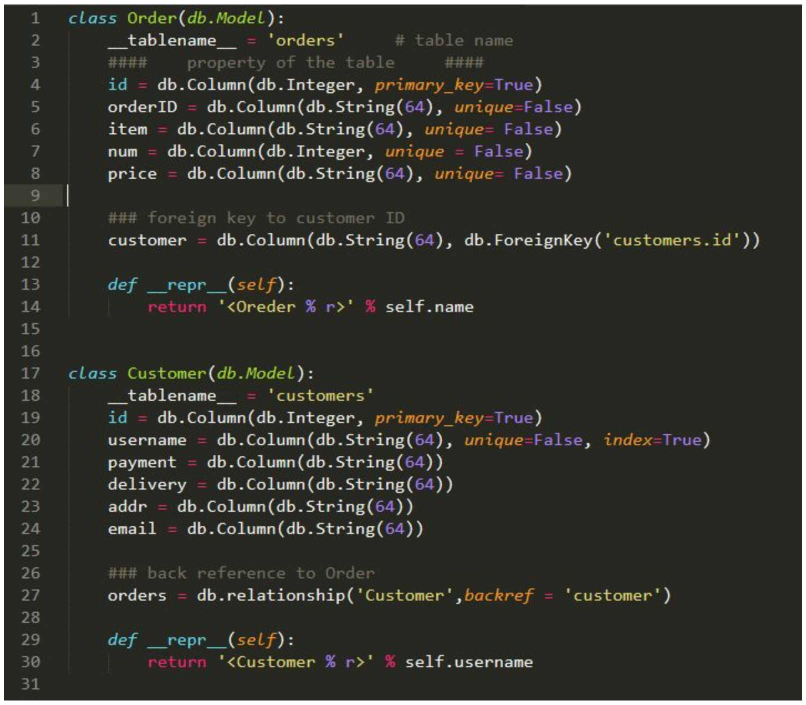
First, as shown in line 3-4 in Figure 3.2.2, Wendy's server will check the validation of the JSON file. If it is invalid, the server will abort 400 error (Bad Request
). Otherwise, the server will process the request and send back a 201 (Created) response. When Wendy's server processes orders, there are four main steps:

1) Create a unique id of Wendy's database (line 7-8 in Figure 3.2.2) and generate a new order for Wendy's database (line 10-15 in Figure 3.2.2) The new order for John's example can be seen in Figure 3.2.4, the value of "id" is id in Wendy's database and the value of "orderID" is id in Shopify.

2) Append a new record in daily record text file (line 18 in Figure 3.2.2). The daily record is designed in case error happens when saving orders into database, it works as a backup. Besides, in real business environment, shop owners may not know how to use SQL search in terminal if they need to check details of some failed orders, so they can check the daily record if needed.

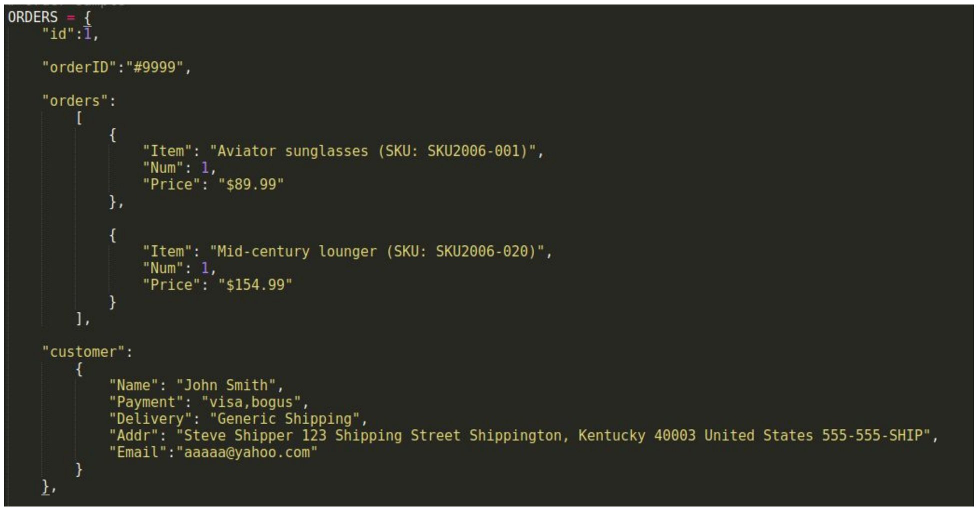
3) Create a query in Wendy's database (line 21 - 47 in Figure 3.2.2). Line 21-31 deals with customer information and line 34- 44 deals with the customer's orders. Wendy's database is designed to use a foreign key to connect customer and his order (line 11 and line 27 in Figure 3.2.3). Line 43 submits the new query into database to finish John's order placement.

4) Send a response to the E2R which includes request status code (201), order IDs in Wendy and in Shopify, customer information and order details. The order part without status code can be seen in Figure 3.2.4.



```
1   class Order(db.Model):
2       __tablename__ = 'orders'      # table name
3       ####    property of the table     ####
4       id = db.Column(db.Integer, primary_key=True)
5       orderID = db.Column(db.String(64), unique=False)
6       item = db.Column(db.String(64), unique= False)
7       num = db.Column(db.Integer, unique = False)
8       price = db.Column(db.String(64), unique= False)
9   
10      ### foreign key to customer ID
11      customer = db.Column(db.String(64), db.ForeignKey('customers.id'))
12  
13      def __repr__(self):
14          return '<Oreder % r>' % self.name
15  
16  
17  class Customer(db.Model):
18      __tablename__ = 'customers'
19      id = db.Column(db.Integer, primary_key=True)
20      username = db.Column(db.String(64), unique=False, index=True)
21      payment = db.Column(db.String(64))
22      delivery = db.Column(db.String(64))
23      addr = db.Column(db.String(64))
24      email = db.Column(db.String(64))
25  
26      ### back reference to Order
27      orders = db.relationship('Customer',backref = 'customer')
28  
29      def __repr__(self):
30          return '<Customer % r>' % self.username
31  
```

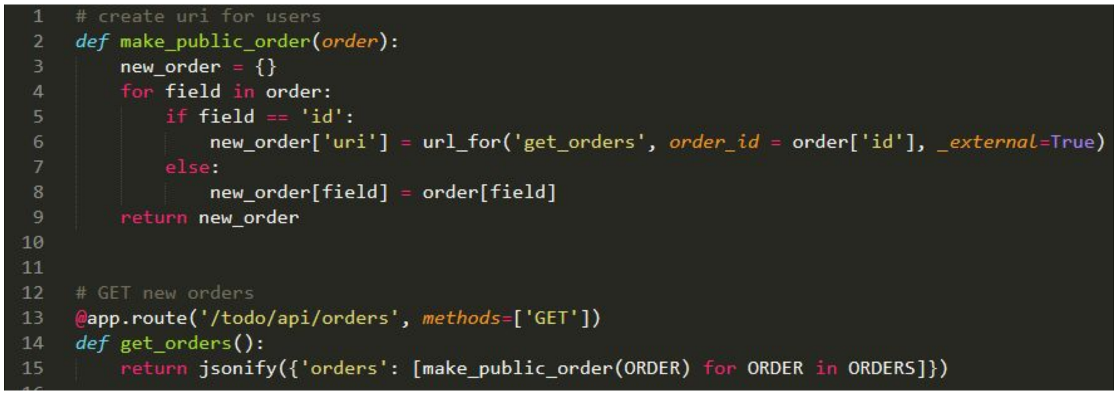
Figure 3.2.3 Database of Customer and Order in Wendy's server



```
ORDERS = {
    "id":1,

    "orderID":"#9999",

    "orders":
        [
            {
                "Item": "Aviator sunglasses (SKU: SKU2006-001)",
                "Num": 1,
                "Price": "$89.99"
            },

            {
                "Item": "Mid-century lounger (SKU: SKU2006-020)",
                "Num": 1,
                "Price": "$154.99"
            }
        ],

    "customer":
        {
            "Name": "John Smith",
            "Payment": "visa,bogus",
            "Delivery": "Generic Shipping",
            "Addr": "Steve Shipper 123 Shipping Street Shippington, Kentucky 40003 United States 555-555-SHIP",
            "Email":"aaaaa@yahoo.com"
        }
    },
```

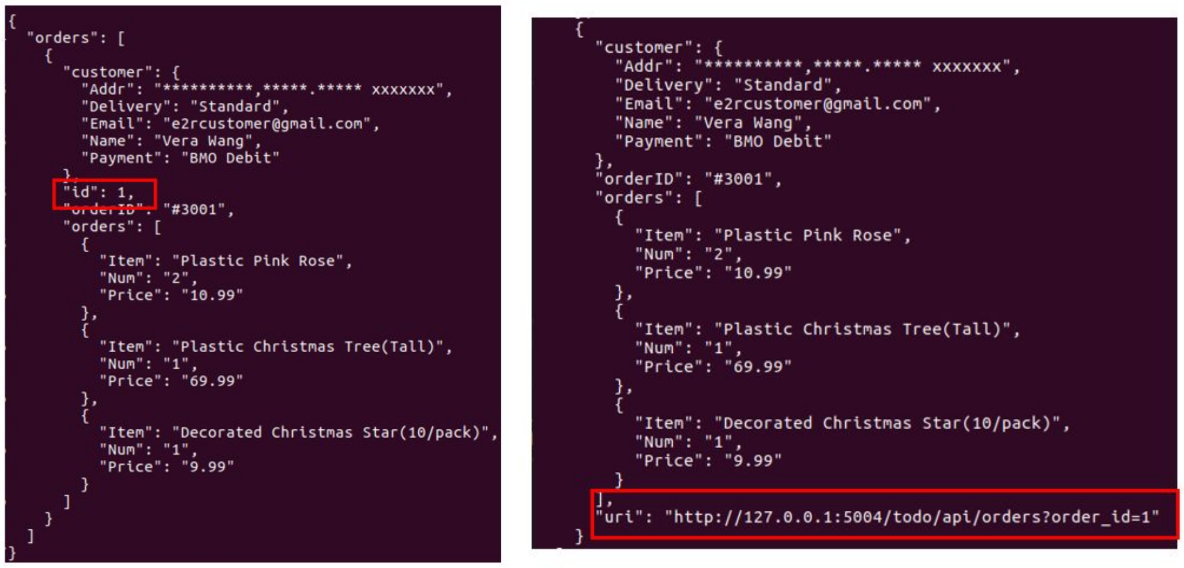
Figure 3.2.4 After Wendy's server process John Smith's order, the order added a Wendy's unique ID will be sent back to the E2R with a 201 status code. The value of "id" is id in Wendy's database and the value of "orderID" is id in Shopify.

If a remote client make a GET call, Wendy's server will reply all orders in current session. This function is designed for the administrator at the E2R end to check order placement status. The detailed process will be discussed in Section 3.3. It is worthy to mention that in newest version, there is a built-in function (make_public_order(), line 2-9 in Figure 3.2.5) in Wendy's server to generate a completed URL to get every single order. That is to say, when the GET method is called in current session, the response will replace Wendy's database order id with a unique URL to call every single order so users do not need to figure out the API URL by themselves. And this function protects Wendy's inner information.



```
 1  # create uri for users
 2  def make_public_order(order):
 3      new_order = {}
 4      for field in order:
 5          if field == 'id':
 6              new_order['uri'] = url_for('get_orders', order_id = order['id'], _external=True)
 7          else:
 8              new_order[field] = order[field]
 9      return new_order
10
11
12  # GET new orders
13  @app.route('/todo/api/orders', methods=['GET'])
14  def get_orders():
15      return jsonify({'orders': [make_public_order(ORDER) for ORDER in ORDERS]})
```

Figure 3.2.5 GET method in Wendy's server



```
{
  "orders": [
    {
      "customer": {
        "Addr": "**********,*****.***** xxxxxxx",
        "Delivery": "Standard",
        "Email": "e2rcustomer@gmail.com",
        "Name": "Vera Wang",
        "Payment": "BMO Debit"
      },
      "id": 1,
      "orderID": "#3001",
      "orders": [
        {
          "Item": "Plastic Pink Rose",
          "Num": "2",
          "Price": "10.99"
        },
        {
          "Item": "Plastic Christmas Tree(Tall)",
          "Num": "1",
          "Price": "69.99"
        },
        {
          "Item": "Decorated Christmas Star(10/pack)",
          "Num": "1",
          "Price": "9.99"
        }
      ]
    }
  ]
}
```

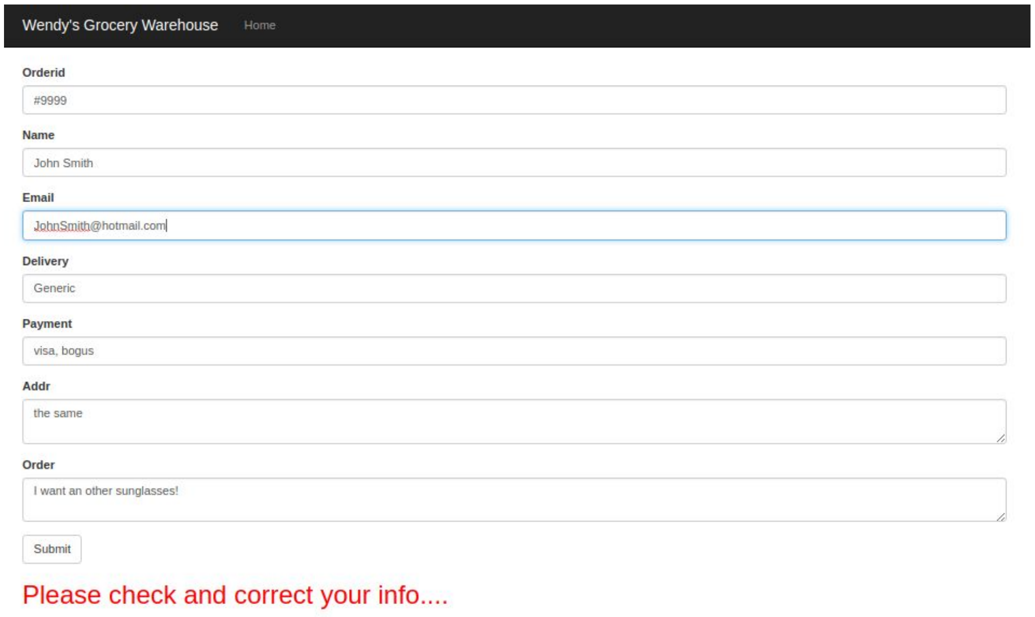
```
{
  "customer": {
    "Addr": "**********,*****.***** xxxxxxx",
    "Delivery": "Standard",
    "Email": "e2rcustomer@gmail.com",
    "Name": "Vera Wang",
    "Payment": "BMO Debit"
  },
  "orderID": "#3001",
  "orders": [
    {
      "Item": "Plastic Pink Rose",
      "Num": "2",
      "Price": "10.99"
    },
    {
      "Item": "Plastic Christmas Tree(Tall)",
      "Num": "1",
      "Price": "69.99"
    },
    {
      "Item": "Decorated Christmas Star(10/pack)",
      "Num": "1",
      "Price": "9.99"
    }
  ],
  "uri": "http://127.0.0.1:5004/todo/api/orders?order_id=1"
}
```

(A) Old version GET response **(B)** Newest version GET response

Figure 3.2.5 Wendy's server's new version reply API URL instead of the raw id in Wendy's database

As for PUT, it helps to update orders in Wendy's database. And DELETE helps to delete placed orders query. However, the two interface are not visible to external servers in consideration of database safety.

Besides, if customer wants to change or cancel their order after the order has been placed, there is an order update interface built in Wendy's server (the interface is in Figure 2.1.6). After customer receive the confirmation Email after Wendy's server already put order query into database, customer can click “Change” button in confirmation Email and be redirected to Shop owner's order update page. For example, if John smith receives his confirmation Email and want to buy one more “sunglasses”, he clicks “Change” and be redirect to “WendyURL.com/ change”. Then, he filled in the form on the “change” interface shown in Figure 3.2.6. When John finishes the form and click “submit”, Wendy's server will check the validation of his updates information and send it to Wendy as Email, and he will see a notice “Order Updated!”.


Wendy's Grocery Warehouse Home

Orderid

#9999

Name

John Smith

Email

JohnSmith@hotmail.com

Delivery

Generic

Payment

visa, bogus

Addr

the same

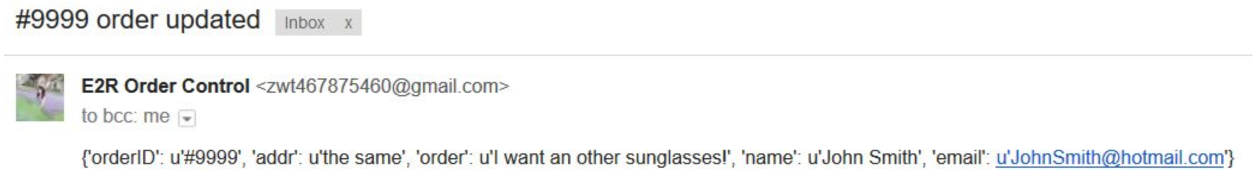
Order

I want an other sunglasses!

Submit

Please check and correct your info....

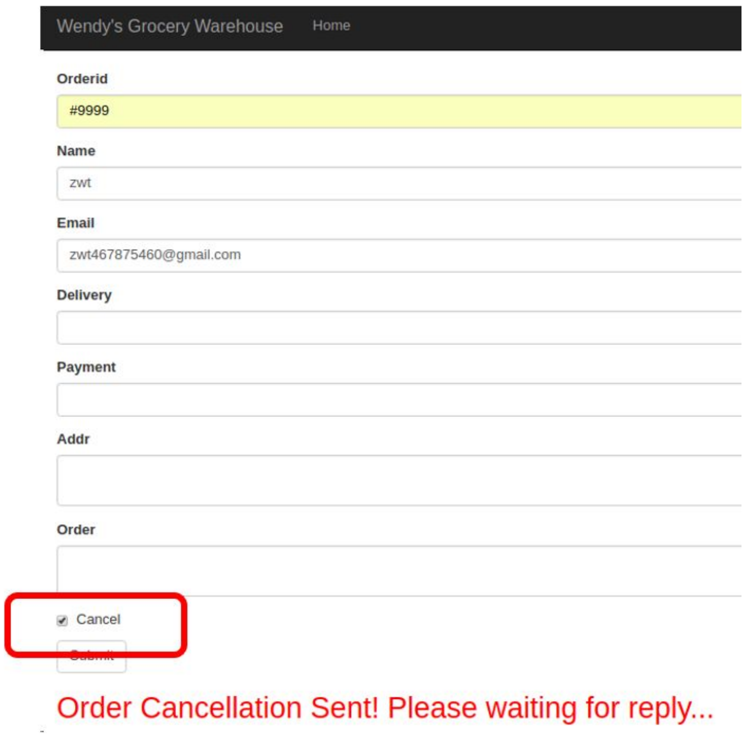
Figure 3.2.6 John Smith's update form


#9999 order updated Inbox x

E2R Order Control <zwt467875460@gmail.com>
to bcc: me

```
{'orderID': u'#9999', 'addr': u'the same', 'order': u'I want an other sunglasses!', 'name': u'John Smith', 'email': u'JohnSmith@hotmail.com'}
```

Figure 3.2.7 Updates report to Wendy



Wendy's Grocery Warehouse Home

Orderid: #9999

Name: zwt

Email: zwt467875460@gmail.com

Delivery:

Payment:

Addr:

Order:

☑ Cancel

Submit

Order Cancellation Sent! Please waiting for reply...

Figure 3.2.8 Cancellation form

If one customer needs to make their order cancelled, he needs to fill in their “Order ID”, “Name” and “Email”, then checks the “cancel” box as shown in Figure 3.2.8. A summary image of Wendy’s server’s functions can be seen in Figure 3.2.9.

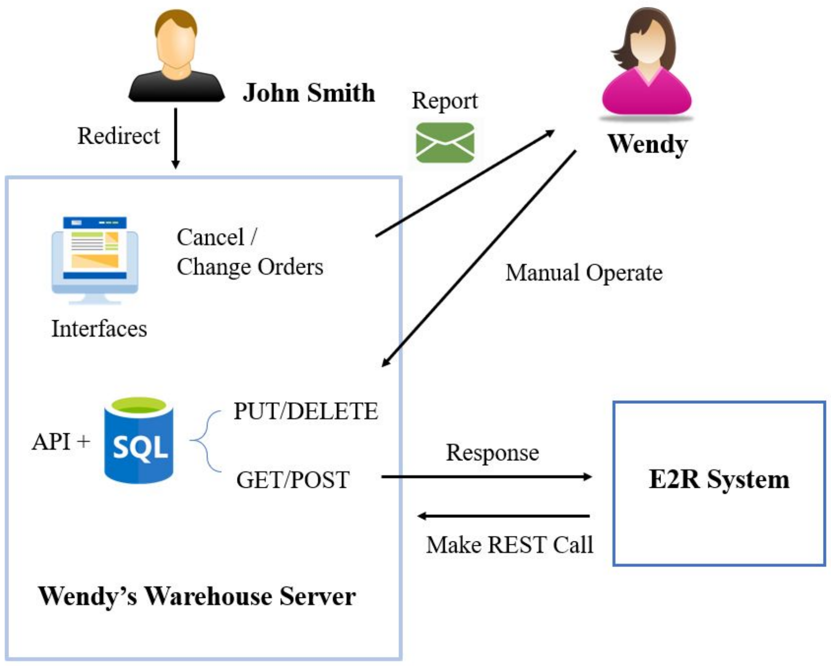
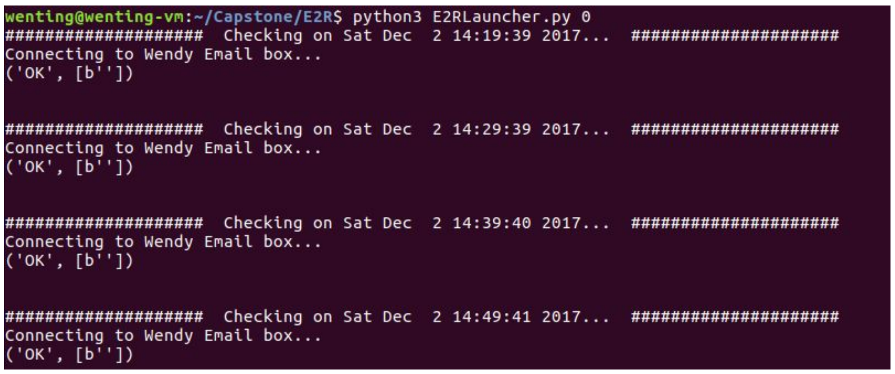


Figure 3.2.9 Wendy’s server’s function summary

3.3 The E2R system

There are 2 modes in the E2R system: test mode and monitor mode. As for test mode, the E2R will poll and process all unread emails in Wendy's mailbox and wait for following commands. As for monitor mode, the E2R will check Wendy's mailbox as in test mode [7]. Afterward, the E2R will wait 10 minutes and check Wendy's mailbox again (shown in Figure 3.3.1). Because the actual working principle for the two modes are the same, for readers to understand the E2R working principle better, this report will only talk about the test mode.



```
wenting@wenting-vm:~/Capstone/E2R$ python3 E2RLauncher.py 0
####################  Checking on Sat Dec  2 14:19:39 2017...  ####################
Connecting to Wendy Email box...
('OK', [b''])

####################  Checking on Sat Dec  2 14:29:39 2017...  ####################
Connecting to Wendy Email box...
('OK', [b''])

####################  Checking on Sat Dec  2 14:39:40 2017...  ####################
Connecting to Wendy Email box...
('OK', [b''])

####################  Checking on Sat Dec  2 14:49:41 2017...  ####################
Connecting to Wendy Email box...
('OK', [b''])
```

Figure 3.3.1 the E2R works on mode 0 to monitor Wendy's server every 10 minutes

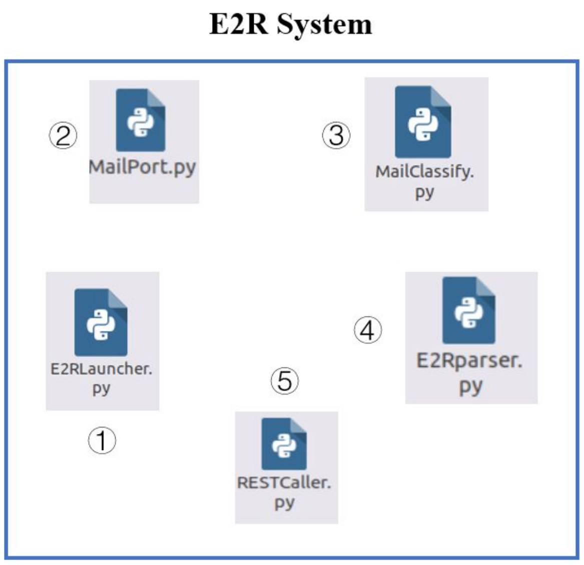


Figure 3.3.2 Five main programs in the E2R system

In the E2R, there are 5 main programs to process Emails and make REST call (Figure 3.3.2). Firstly, "E2RLauncher.py" (Figure 3.3.2 ①) is used to launch the the E2R server as the main switch (Figure 3.3.3 (1)). The launcher will then trigger "MailPort.py" to poll Emails from Wendy's mailbox (Figure 3.3.3 (2)), and there are two working modes to choose from (test mode or monitor mode) when launching the E2R system. For example, if running on test mode, when the E2R just finishes processing John Smith's order, it will block and ask administrator what to do next (Figure 3.3.4). If user reply "YES", the E2R system will reply all orders in current session by making a REST call (method = "GET") to Wendy's server, the "GET" call will trigger "get_orders()" function in Wendy's server as discussed in Section 3.2. If the user replies "Terminate", the E2R server will shut down. Else, the E2R will check Wendy's mailbox again.

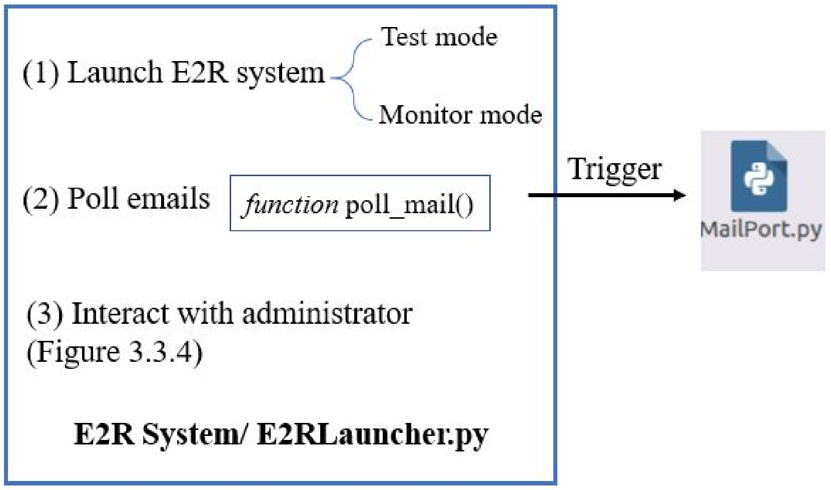
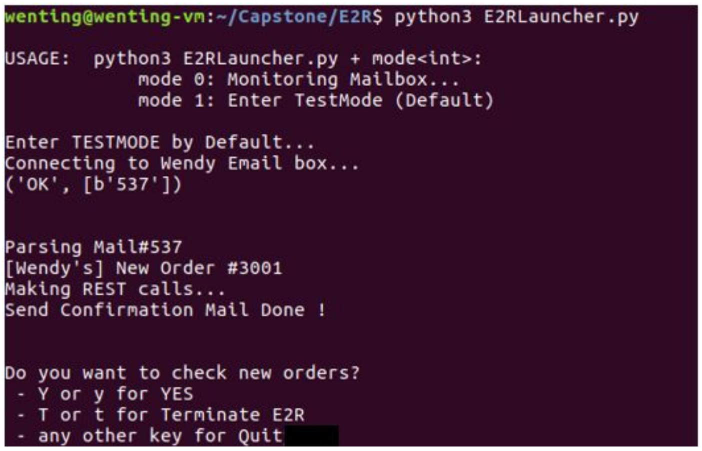


Figure 3.3.3 "E2RLauncher.py" functions outline



```
wenting@wenting-vm:~/Capstone/E2R$ python3 E2RLauncher.py

USAGE:  python3 E2RLauncher.py + mode<int>:
            mode 0: Monitoring Mailbox...
            mode 1: Enter TestMode (Default)

Enter TESTMODE by Default...
Connecting to Wendy Email box...
('OK', [b'537'])


Parsing Mail#537
[Wendy's] New Order #3001
Making REST calls...
Send Confirmation Mail Done !


Do you want to check new orders?
 - Y or y for YES
 - T or t for Terminate E2R
 - any other key for Quit
```

Figure 3.3.4 the E2R ask administrator what to do next

Secondly, "MailPort.py" (② in Figure 3.3.2) is a module in control of polling and sending emails (Figure 3.3.5). Assume the data stream is a river, emails are boats which carry data from end to end, the "MailPort.py" works as a port of the E2R system to get and send emails. On the one hand, "MailPort.py" helps to poll emails (Figure 3.3.5 ①). As mentioned in last paragraph, when the E2R server starts running, “MailPort.py” will be triggered by “the E2RLauncher.py” to poll emails from Wendy’s mailbox by method “poll_mail()”. The “poll_mail()” method uses “ReceiveMailDealer” class to finish its job. This class is based on IMAP protocol and refer to pyMail from Github [8]. To poll emails, “ReceiveMailDealer” will first scan Wendy’s mailbox for unread mails and return unread mails as mail objects, then a method called “getMailInfo()” in “ReceiveMailDealer” class will extract email subject and body as string and pass them to “MailClassify.py” to finish following work.

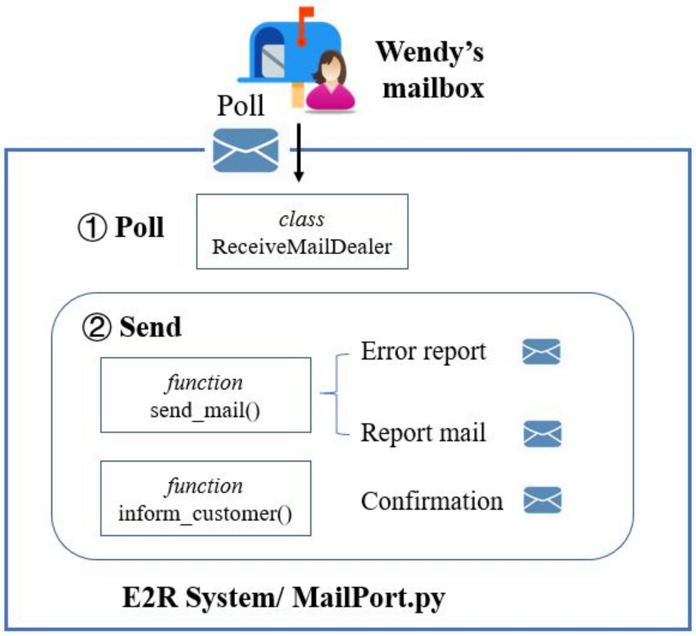
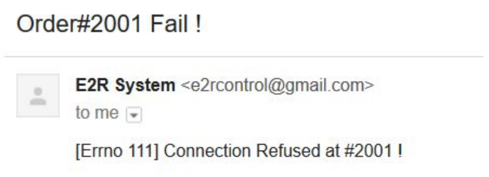


Figure 3.3.5 “MailPort.py” functions outline

On the other hand, as for sending mails in the E2R, three kinds of mails needs to be sent and they are all based on SMTP protocol (Figure 3.3.5 ②). If the order is placed successfully, method “inform_customer()” will be called to send “confirmation mail” to customers and “report mail” to shop owner. For example, when John Smith’s is finished, John’s mail and Wendy’s mail are shown in Figure 2.1.4 and 2.1.5. In Figure 2.1.4, customer can be redirected to Wendy’s server to update their orders by clicking “Change” or “Cancel” button.

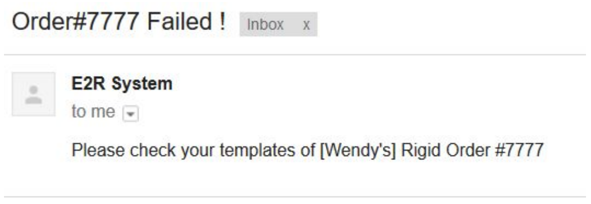
Also, there may be some errors when the E2R is processing orders. At this time, the E2R will send “error report” to Wendy’s mail box. For example, if users in Shopify design a very complicated Email template which the E2R cannot parse, a mail in Figure 3.3.6(a) will be sent to Wendy. If Wendy’s server is shut down accidentally, a mail (Figure 3.3.6(b)) carries the error type and order ID will be sent. At the mean time, the E2R will stop parse following mails and terminate itself in case that some orders may be missed. Sometimes there maybe other errors, the E2R will alse send the error type with mistake order id to shop owner’s mailbox. For example, a new programr of Wendy's server delete database by mistake, Wendy may get a report Email shown in Figure 3.3.6(c). As can be seen in Figure 3.3.7, by using the E2R system to process Emails, the shop owner will finish get a list of order status with neat details in their mailbox list.


Order#2001 Fail !

E2R System <e2rcontrol@gmail.com>
to me

[Errno 111] Connection Refused at #2001 !

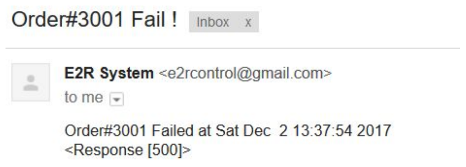
(a) Connection Error


Order#7777 Failed ! Inbox x

E2R System
to me

Please check your templates of [Wendy's] Rigid Order #7777

(b) Template Error

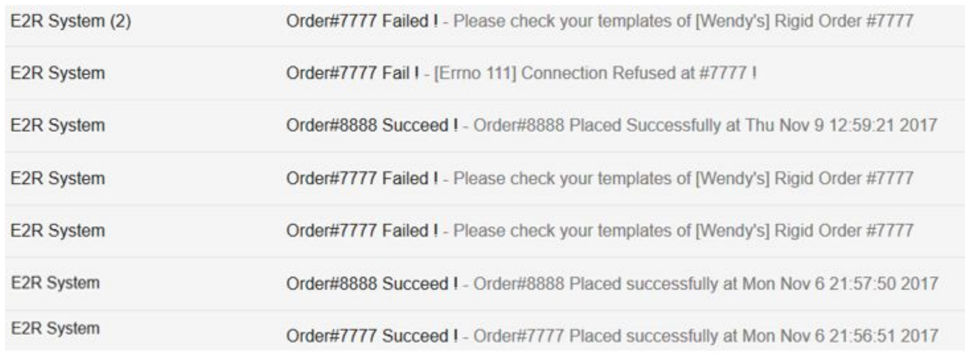

Order#3001 Fail ! Inbox x

E2R System <e2rcontrol@gmail.com>
to me

Order#3001 Failed at Sat Dec 2 13:37:54 2017
<Response [500]>

(c) Internal Server Error (HTTP Code 500)

Figure 3.3.6 Error report emails



E2R System (2)	Order#7777 Failed ! - Please check your templates of [Wendy's] Rigid Order #7777
E2R System	Order#7777 Fail ! - [Errno 111] Connection Refused at #7777 !
E2R System	Order#8888 Succeed ! - Order#8888 Placed Successfully at Thu Nov 9 12:59:21 2017
E2R System	Order#7777 Failed ! - Please check your templates of [Wendy's] Rigid Order #7777
E2R System	Order#7777 Failed ! - Please check your templates of [Wendy's] Rigid Order #7777
E2R System	Order#8888 Succeed ! - Order#8888 Placed successfully at Mon Nov 6 21:57:50 2017
E2R System	Order#7777 Succeed ! - Order#7777 Placed successfully at Mon Nov 6 21:56:51 2017

Figure 3.3.7 the E2R mail status report list in Wendy's mailbox

Thirdly, "MailClassify.py" (Figure 3.3.2 ③) classifies Emails extracted by "MailPort.py" into three kinds by file type: Text Email, HTML Email and JSON Email. In fourth step, the three kinds of Email will b parsed by three methods in "the E2RParser.py" (Figure 3.3.2 ④) —— "generate_order(mail,subj)", "DIY_generate_order(mail,subj)" and "simple_generate_order(mail,subj)" separately. If parsed successfully, the three methods will return a dictionary contains customer information and order details. The dictionary will then be passed to "RESTCaller.py" (Figure 3.3.2 ⑤). This process can be summarized to three parts: classification, translation and make REST call. A completed working flowchart of the three parts can be seen in Figure 3.3.8.

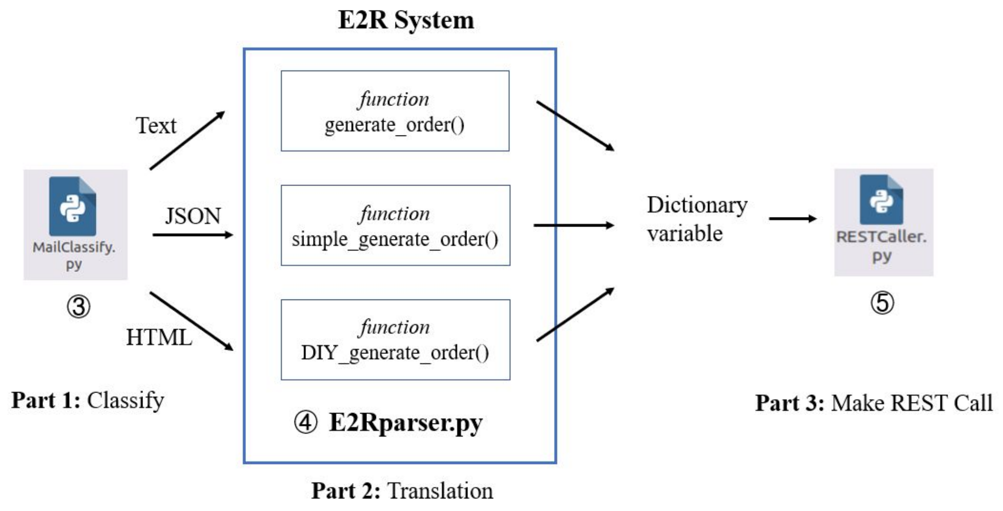


Figure 3.3.8 “MailClassify.py”, “the E2Rparser.py”, “RESTCaller.py” working flowchart

Last, in “RESTCaller.py”, the dictionary variable will be encoded to JSON and be carried to make a REST call to Wendy’s server (Figure 3.3.9 (A)). After making the call, the E2R will block and wait for Wendy’s response. If the response code is 201 (Created) , the E2R “RESTCaller.py” will trigger “MailPort.py” to send successful status report to Wendy and a confirmation Email to customer John Smith (Figure 3.3.9 (B)). John Smith’s Email Address is extracted in step 3 in “the E2RParser.py”. What’s more, after finish one session (check all unread Emails in Wendy’s mailbox), as mentioned in Section 3.2, the E2R will ask whether administrator would like see all new orders. To get all new orders in the session, the E2R will make a REST (method = GET) to Wendy’s server to get the information. This REST is also made by “RESTCaller.py”.

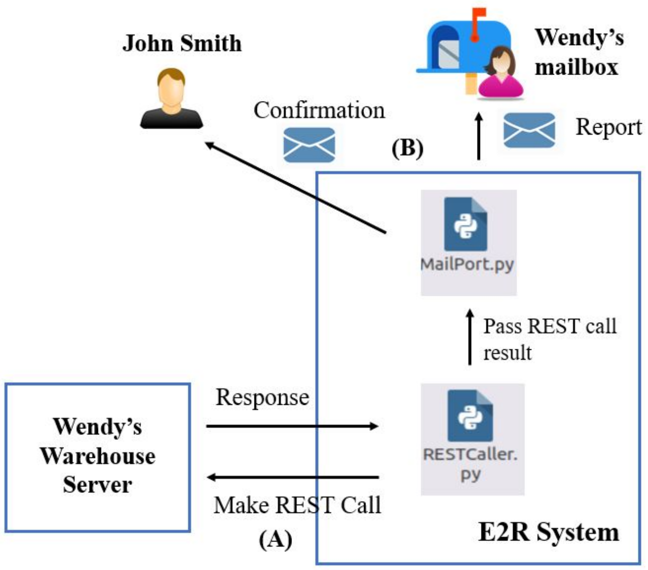


Figure 3.3.9 "RESTCaller.py" interacts with Wendy's server and "MailPort.py" send emails to users

3.4 Systems Interactions

After knowing how the three main systems works separately, let us look back and make a conclusion of how Shopify, the E2R, and Wendy's server work together. A summary is in Figure 3.4.1.

First of all, John Smith makes an order on Shopify and Shopify sends order summary to Wendy's mailbox. The process is simulated by "Shopify/SimulatedWeb" in the capstone project. Then, the E2R translates Emails in Wendy's mailbox to JSON to make REST calls to Wendy's server, these processes are finished by 5 programs in the E2R. MailPort.py polls Emails from Wendy's mailbox timely, passes extracted Emails to MailClassify.py to classify, MailClassify.py passes classified Email to 3 different parsers in the E2RParser.py to get translated JSON file, the E2RParser.py passes the JSON file to RESTCaller.py to make the REST Call. In Wendy's server, when the server receives REST call from the E2R, if it is a POST call, Wendy's server will save the information carried in the call into database and reply a 201 status code. if it is a GET call, Wendy's server will reply the asked data with a 200 status code. After receiving response from Wendy's server, the E2R will send order status report to Wendy's mailbox. If the order is successful, the E2R will also send a confirmation Email to John Smith. After John receives the Email, he can make changes or cancel proposal of his order by clicking the link in the confirmation Email. At last, the changes or cancel proposal will be sent to Wendy's mailbox.

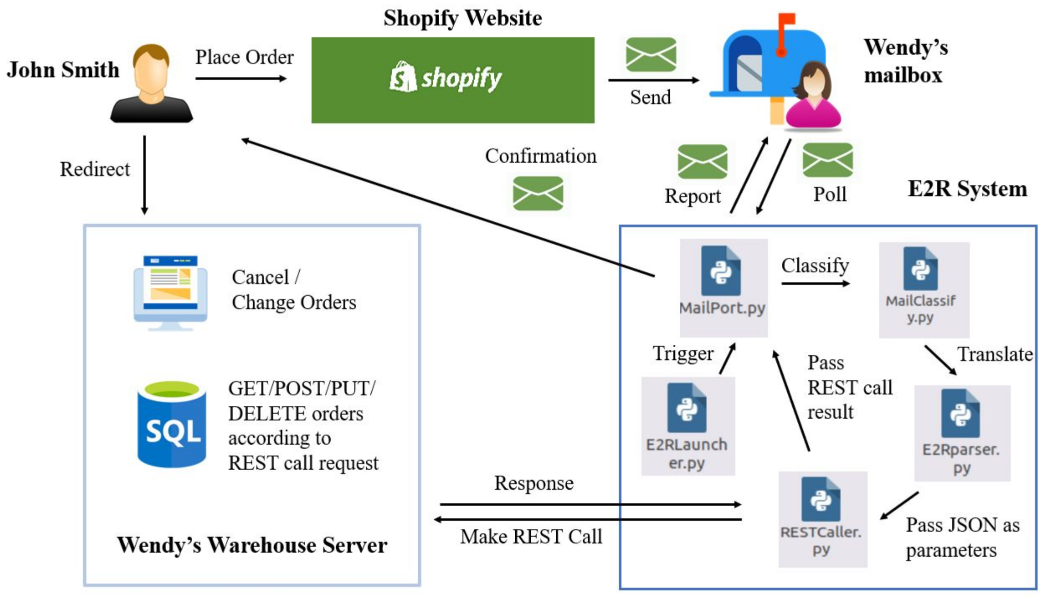


Figure 3.4.1 System interactions summary

4. Translations

Translating Email into rigid JSON file is the main function and initial design propus of the E2R system. According to today's mail market, there are 2 widely-used Email type: text and HTML. Besides, because of the user template interface design of Shopify platform, it is very convenient to use a rigid template (send JSON in Email directly). This Section will introduce the 3 kinds of Emails' translations.

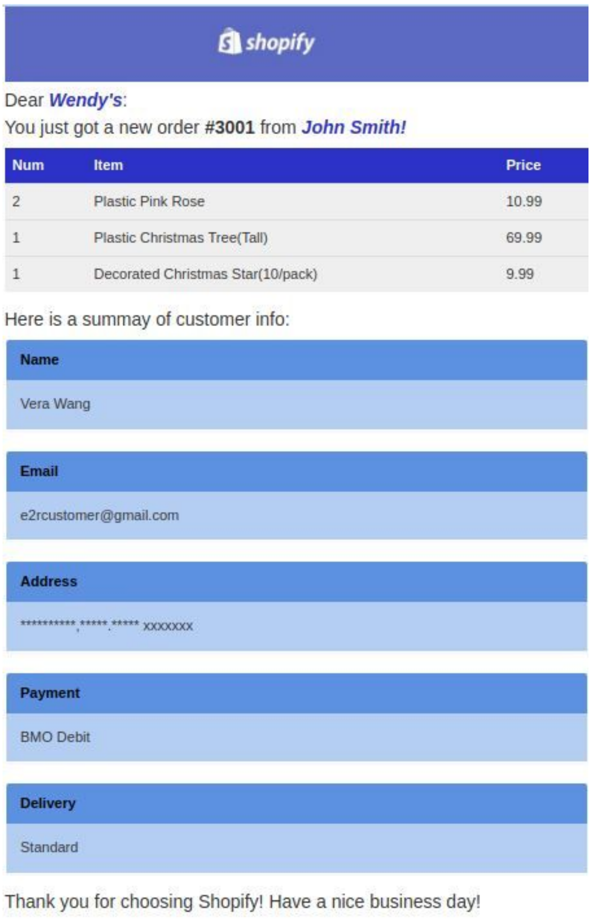
After Emails have been polled from Wendy's mailbox, the E2R MailPort will pass two string variables (subject and body) to the E2R MailClassify. The "MailClassify.py" program will classify Emails by its subject. If the subject matches one of the three main Email subjects format, the mail will be translated. Else, the mail will be skipped as a spam mail.

4.1 Default Email Template (Text)

If one Email is classified as Default template, the subject and body string of it will be passed to "the E2RParse.py" to be parsed be method "generate_order(subj,body)". A sample default email template is shown in Figure 1.1.2(a). According to the content permutation and data property, the body string will be traversed by a for loop line by line to extract key words. For example, in the default template, there is a "$" before the price number. So the program can use "$" to locate item details and it's exact price. As another example, the "Payment processing method", "Delivery method" and "Shipping address" of one order show up sequentially at last of the default template. So when "Payment processing method" is detected in one line, the program will append the line content every other line into an array. The program does append operation every other line because it needs to skip the content subtitle (e.g. "Delivery method").

4.2 User-Defined Email Template (HTML)

If one Email is classified as HTML template, the subject and body string of it will be passed to "the E2RParse.py" to be parsed be method "generate_DIY_order(subj,body)". The method "generate_DIY_order(subj,body)" use a library *Beautiful Soup4(refer to as "bs4" below)* [11] to parse HTML file by checking HTML tags. A sample HTML Email of John Smith is shown in Figure 4.2.1. Part of the HTML code of the Email is shown in Figure 4.2.2.



Dear ***Wendy's***:
You just got a new order **#3001** from ***John Smith!***

Num	Item	Price
2	Plastic Pink Rose	10.99
1	Plastic Christmas Tree(Tall)	69.99
1	Decorated Christmas Star(10/pack)	9.99

Here is a summay of customer info:

Name
Vera Wang

Email
e2rcustomer@gmail.com

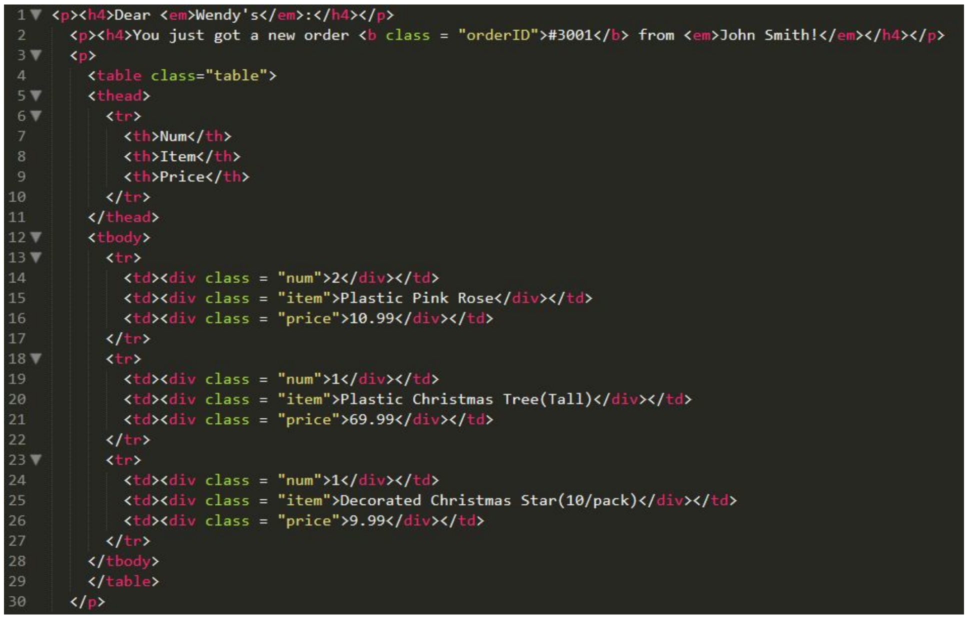
Address
**********,*****,***** xxxxxxx

Payment
BMO Debit

Delivery
Standard

Thank you for choosing Shopify! Have a nice business day!

Figure 4.2.1 A sample HTML Email of John Smith



```
1  <p><h4>Dear <em>Wendy's</em>:</h4></p>
2    <p><h4>You just got a new order <b class = "orderID">#3001</b> from <em>John Smith!</em></h4></p>
3    <p>
4      <table class="table">
5      <thead>
6        <tr>
7          <th>Num</th>
8          <th>Item</th>
9          <th>Price</th>
10       </tr>
11     </thead>
12     <tbody>
13       <tr>
14         <td><div class = "num">2</div></td>
15         <td><div class = "item">Plastic Pink Rose</div></td>
16         <td><div class = "price">10.99</div></td>
17       </tr>
18       <tr>
19         <td><div class = "num">1</div></td>
20         <td><div class = "item">Plastic Christmas Tree(Tall)</div></td>
21         <td><div class = "price">69.99</div></td>
22       </tr>
23       <tr>
24         <td><div class = "num">1</div></td>
25         <td><div class = "item">Decorated Christmas Star(10/pack)</div></td>
26         <td><div class = "price">9.99</div></td>
27       </tr>
28     </tbody>
29     </table>
30   </p>
```

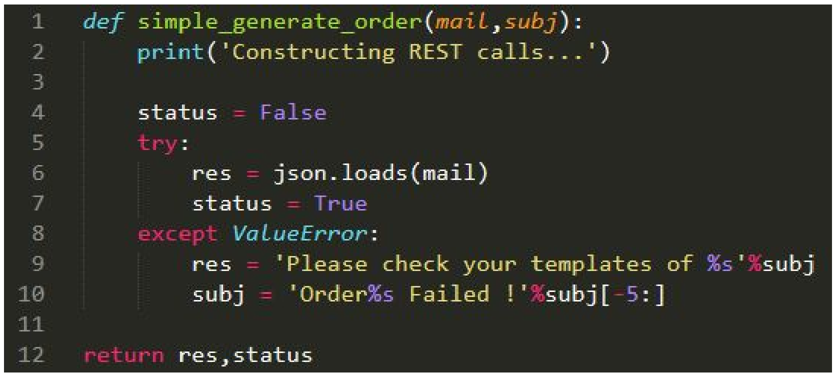
Figure 4.2.2 First half (contains order details and order ID) of John's Order Email HTML code

Some wrapped function of bs4 can traverse a whole HTML file and return raw text nested in specific HTML tag. As can be seen in Figure 4.2.2 line 2, there is a pair of HTML tag <b></b> with John's order ID nested in it. In the opening tag <b>, it has a class "orderID". Function "generate_DIY_order(subj,body)" call a method "find("b", attrs = {"class" : "orderID"})" in bs4 to find the specific tag "<b class = "orderID">" and return the raw text nested in it.

By using this parsing method, the permutation of information and data property of the contents inside targeted tags do not matter any more. So, only if shop owners design their HTML template with the right HTML tag, right messages can be extracted regardless of how the HTML structure really is.

4.3 Rigid Email Template (JSON)

Because Shopify provides shop owners with the feature to modify their Email templates freely, it is feasible to send Email which is already structured in JSON directly, so the E2R do not need to do the parse again. In the situation of rigid template, when MailClassify pass the JSON like string to the E2Rparser, a method "simply_generate_order(subj,body)" will be used to check the validation of the string. In the method, a python built-in method "json.loads()" will be called to encode the string to JSON (Line 6 in Figure 4.3.1). If the operation fails, which means the JSON structure is wrong (in most case, it is because of template designer's typo), a template error report will be sent to Wendy. And the status will be set to False, which will inform following program, the parse process has failed so do not make REST calls. On the other hand, if the encode succeeds, the status will be set to True. This procedure is shown in Figure 4.3.2.



```python
def simple_generate_order(mail,subj):
    print('Constructing REST calls...')

    status = False
    try:
        res = json.loads(mail)
        status = True
    except ValueError:
        res = 'Please check your templates of %s'%subj
        subj = 'Order%s Failed !'%subj[-5:]

return res,status
```

Figure 4.3.1 Method simple_generate_order in “the E2RParser.py”

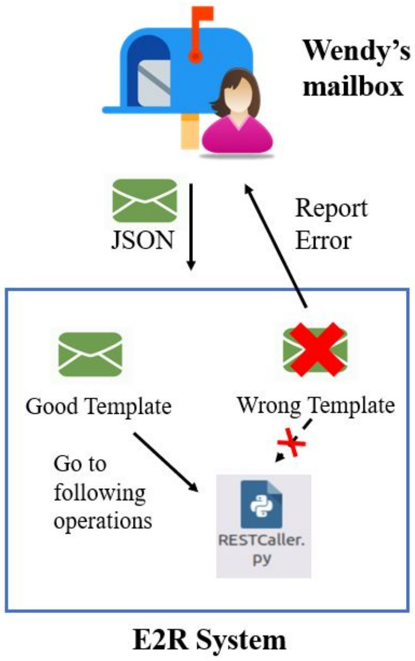
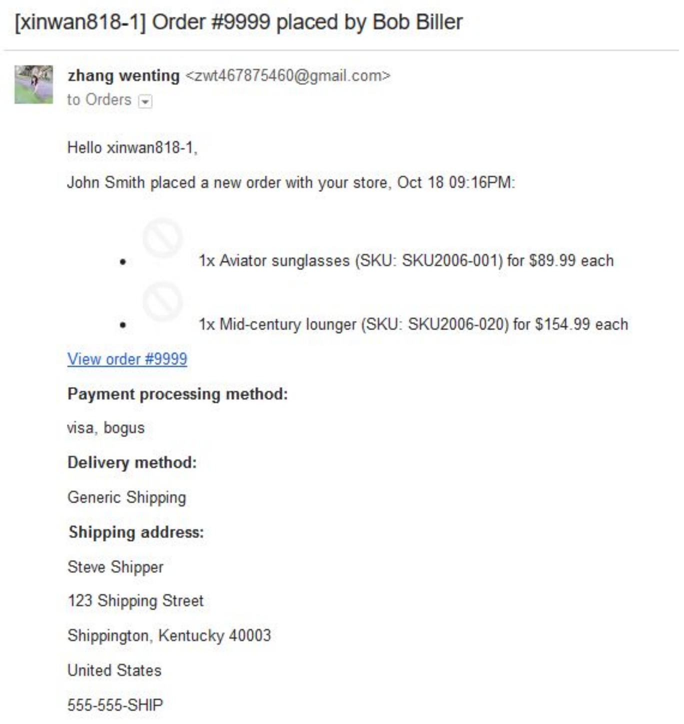


Figure 4.3.2 the E2R deals with good/wrong JSON like Emails

5. Testcase Groups

5.1 Templates

the E2R system is tested by one kind of “Text” template (Figure 5.1.1), three kinds of HTML template (Figure 5.1.2) [10] and one JSON like template (Figure 5.1.3). The “Text” template is the default template that Shopify defined for shop owners and can be found and modify at Shopify users' "settings" panel. Four different "Text" Emails are used to do the test and get good output. The JSON like template is designed to match the database query structure of Wendy's server. Four different JSON like Emails are used to do the test and get good output. As for the HTML template, the overall template designs of every single HTML template are different, while in every design, the HTML tag of key information is the same so data can be extracted correctly. For every design, two Emails are used to test the E2R system and the (2*3 = 6 in total) templates works fine. In summary, 14 Email testcases are used to test the E2R system and the the E2R system works for all 14 test cases.



[xinwan818-1] Order #9999 placed by Bob Biller

zhang wenting <zwt467875460@gmail.com>
to Orders

Hello xinwan818-1,

John Smith placed a new order with your store, Oct 18 09:16PM:

- 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

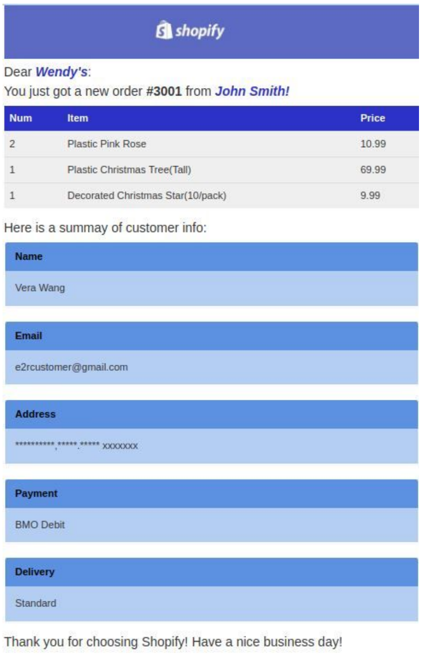
123 Shipping Street

Shippington, Kentucky 40003

United States

555-555-SHIP

Figure 5.1.1 Template 1: Default



Dear ***Wendy's***:
You just got a new order **#3001** from ***John Smith!***

Num	Item	Price
2	Plastic Pink Rose	10.99
1	Plastic Christmas Tree(Tall)	69.99
1	Decorated Christmas Star(10/pack)	9.99

Here is a summay of customer info:

Name
Vera Wang

Email
e2rcustomer@gmail.com

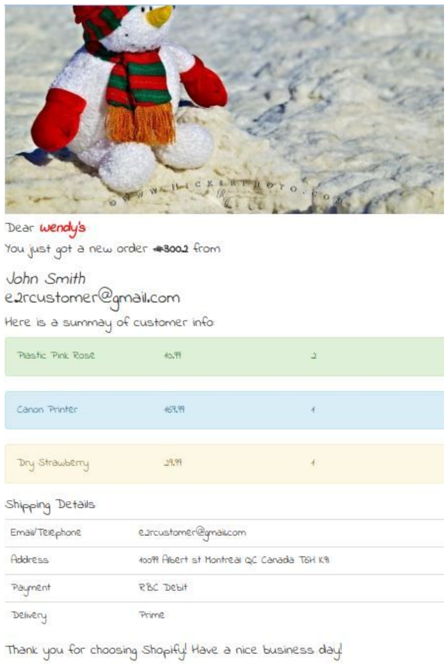
Address
**********,*****,***** xxxxxxx

Payment
BMO Debit

Delivery
Standard

Thank you for choosing Shopify! Have a nice business day!

(A) Template 2: HTML-A



Dear **wendy's**
You just got a new order **#3002** from
John Smith
e2rcustomer@gmail.com
Here is a summay of customer info

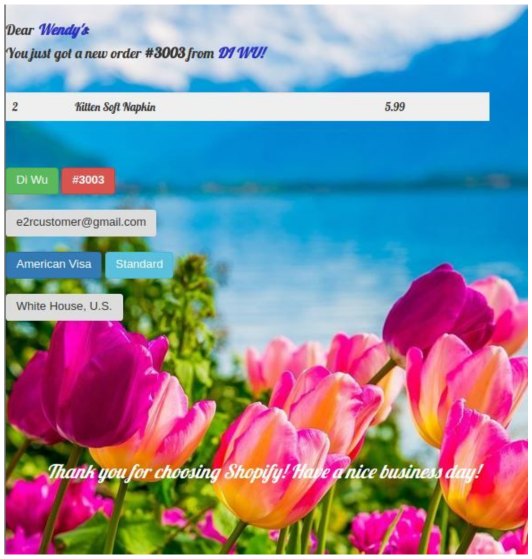
Plastic Pink Rose	10.99	2
Canon Printer	469.99	1
Dry Strawberry	29.99	1

Shipping Details

Email/Telephone	e2rcustomer@gmail.com
Address	10099 Albert st Montreal QC Canada T6H K9
Payment	RBC Debit
Delivery	Prime

Thank you for choosing Shopify! Have a nice business day!

(B) Template 3: HTML-B



Dear Wendy's
You just got a new order #3003 from DI WU!

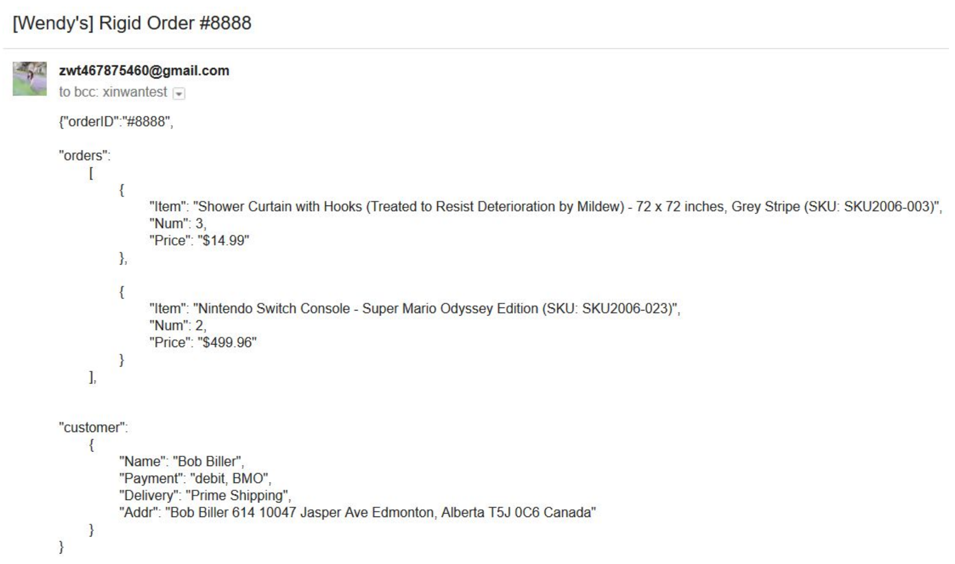
2	*Kitten Soft Napkin*	5.99

Di Wu #3003
e2rcustomer@gmail.com
American Visa Standard
White House, U.S.

Thank you for choosing Shopify! Have a nice business day!

(c) Template 4: HTML-C

Figure 5.1.2 HTML templates (A, B, C)


[Wendy's] Rigid Order #8888

zwt467875460@gmail.com
to bcc: xinwantest

```
{"orderID":"#8888",

"orders":
    [
        {
            "Item": "Shower Curtain with Hooks (Treated to Resist Deterioration by Mildew) - 72 x 72 inches, Grey Stripe (SKU: SKU2006-003)",
            "Num": 3,
            "Price": "$14.99"
        },

        {
            "Item": "Nintendo Switch Console - Super Mario Odyssey Edition (SKU: SKU2006-023)",
            "Num": 2,
            "Price": "$499.96"
        }
    ],

"customer":
    {
        "Name": "Bob Biller",
        "Payment": "debit, BMO",
        "Delivery": "Prime Shipping",
        "Addr": "Bob Biller 614 10047 Jasper Ave Edmonton, Alberta T5J 0C6 Canada"
    }
}
```

Figure 5.1.3 Template 5: JSON

5.2 Test Programs

Although the simulated website of Shopify works fine to send customer order summary Emails, in practical situation, the E2R system may need to be robust enough to handle thousands of Emails in one day or several Emails coming at the same time. That is to say, the simulated website of Shopify discussed before works as one client end, while we need a program to simulate a bunch of client ends to use Shopify and send mails to the E2R system.

In this case, a test program called "sender.py" is designed to to tests. A module named MailProcessor is used to generate special Emails and send them. The "sender.py" has 5 modes. Users can choose mode to run the program by add a parameter after "sender.py". Functions of the 5 modes are listed in Table 5.2.1.

Mode	Function
0	Send all Emails in every 5 seconds
1	Send default mail 1 2 3 4 and rigid Email 1 2 3 4 and 6 HTML Emails
2	Send 1 DIY Email
3	Send 1 rigid Email
4	Send 1 default Email

6. Conclusion and Future Work

In conclusion, the MINT Capstone Project is mainly implemented under Python 3. There are three systems implemented to simulate and demonstrate the working environment of Email-to-REST User Request Translation System (the E2R system). For the Shopify part, a Django based website is implemented to simulate how Shopify works, and a "sender.py" program and a "MailProcessor" module are implemented to send 14 different test emails separately and together, once and continuously. For Wendy's warehouse server part, a Flask based website is implemented to simulate a small shop's warehouse server which can take 4 kinds of REST calls and make related changes to its database. For the E2R system, 5 main programs are implemented to poll, extract, parse, send emails, make REST calls and report errors.

Generally speaking, the two main functions of the E2R system are communication and translation. Emails are the communication tools between the E2R and human beings. The third party server (Wendy's warehouse server) and the E2R system interact with each other via REST calls. To send and receive emails, the E2R system is designed based on SMTP and IMAP4 protocols. The email server is Gmail. The interaction between the E2R and Wendy's server is based on HTTP protocol and Representational state transfer services. This process uses "requests" API.

In terms of translation, by analysing property of Email templates, the E2R system translate three main kinds of Emails to JSON request. To translate JSON response from third party server (Wendy's warehouse server) back to Email, the E2R use "Jinja 2" API to render Emails [9].

With the help of the E2R, shop owner now do not need bother to check every Email all by their own, while in the future, there is still some work can to be done to make the E2R better. First, although the E2R can handle 3 types of Emails, templates needs to be pre-defined. To make it more convenient to use, natural language library (e.g. NLTK, TextBlob) can be taken into consideration. Besides, there may be some mistakes in customer order summary Emails, which can lead to failure of translation. Therefore, a Q&A system can be added to the E2R system to interact with customer to get the correct information.

Acknowledgement

First of all, I want to express my thankfulness to my instructor and my parents. My instructor Dr. Lu is a really patient and strict teacher, who helps me to be strict to myself not only with the programming but also in related documents (e.g. Dashboard, GitHub document, all reports.). As for my parents, they always encourage me when I am in trouble and provide me with a good environment so I can concentrate on my studies.

Then, I would like to thank myself for never giving up when it is hopeless and always insisting when trapped with bottlenecks. Although the E2R system is only a simple student project, it is my first completed project in Computer Science. The 2221 lines of python codes makes me feel confident in my career.

Last, I really appreciate the time in ATH with Chang and Shan. I will never forget the time when we sit together, focusing on our own studies or enjoying our homemade meals. Hope we can all do well in our future researches and working lives.

Related Links

- Original Approved Proposal:
  https://docs.google.com/document/d/1OhsN6UMBzjNc0WpRmM4TUvibJmAZBC8g4OX4v9QucxQ/edit

- GitHub:
  https://github.com/xinwan818/Email-to-REST-User-Request-Translation-System-

- Templates:
  https://docs.google.com/a/ualberta.ca/presentation/d/1oYkCCSTnax1pI99E5xNwEMQ4jq7KRy574OsQPc252yg/edit?usp=sharing

- User Manual:
  https://docs.google.com/a/ualberta.ca/document/d/1JiuRu-kJ-Kv4NGVFvRA7oV9JIf3q_INrsukXk9-2ZZ0/edit?usp=sharing

- DashBoard:
  https://docs.google.com/document/d/1e50fUk2AZETpWjvnlDUAgh_kcjb1TzmeMvCdV0F_mMI/edit?ts=59de6b1d

References

[1] Mail Parser:
https://mailparser.io/

[2] Julie Desk:
https://www.juliedesk.com/

[3] Architectural Styles and the Design of Network-based Software Architectures, UNIVERSITY OF CALIFORNIA, IRVINE, Roy Thomas Fielding,2000

[4] Flask Development:
https://coddyschool.com/upload/Flask_Web_Development_Developing.pdf

[5] Flask API Design:
https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask

[6] Django Development: https://djangobook.com/

[7] Treading Design: https://docs.python.org/3/library/threading.html

[8] the E2R MailPort Design (PyMail): https://github.com/paramiao/pyMail

[9] Email render(Jinja2): http://jinja.pocoo.org/docs/2.10/

[10] Template Design(Bootstrap3/4): https://www.w3schools.com/bootstrap.com

[11] the E2R HTML parse(Beautifulsoup 4):
https://www.crummy.com/software/BeautifulSoup/bs4/doc/