



IS213

Enterprise Solution Development

AY2018-2019, Term 2

E-Commerce Microservices

Team ID: G6T4

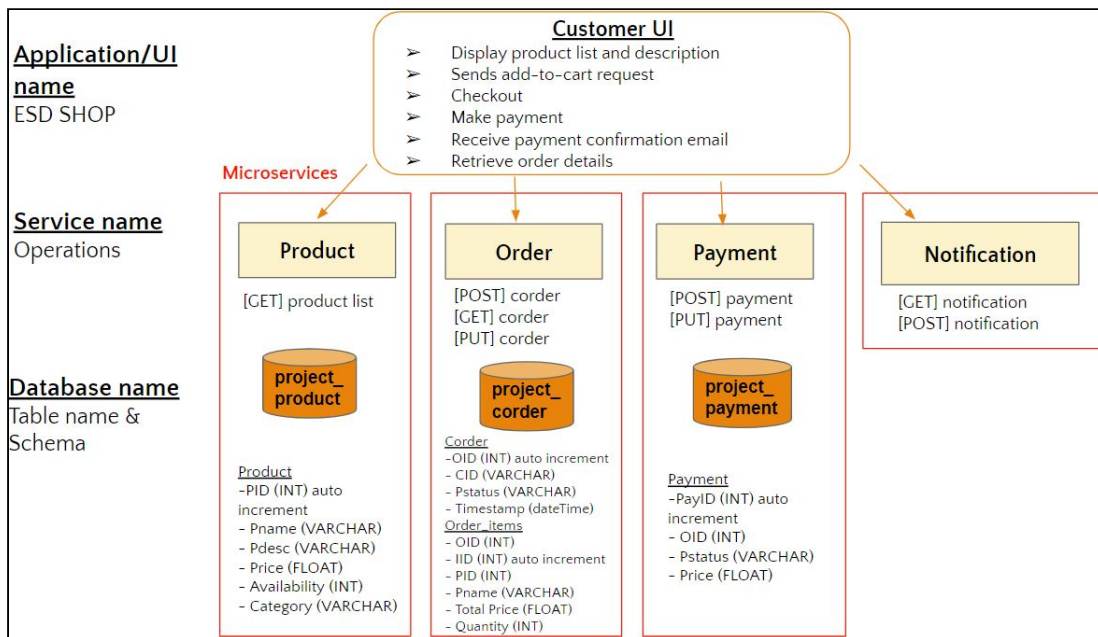
Name(s): Bryan Tan Wei Yoong
Charmaine Tay Shi Yun
Loh Yu Jin
Tan Sok Yi
Teng Jing Wen
Yoon Ju Young

1. Introduction	3
2. Technical Overview Diagram	3
3. User Scenarios	3
3.1 User Scenario 1	3
3.1.1 User searches for products to view descriptions	3
3.1.2 User adds product(s) to cart	3
3.2 User Scenario 2	4
3.2.1 User checks out order	4
3.2.2 User makes payment	4
3.3 User Scenario 3	5
3.3.1 User receives payment confirmation email upon successful payment	5
3.3.2 User tracks order	6
4. Web Services	6
5. Usage of XML schema	7
6. Graphical User Interface	8
7. Going Beyond the Labs	8
7.1 Twitter API	8
7.2 TIBCO For-Loop	8
7.3 PayPal API	8
7.4 Email	8
7.5 Telegram Bot	8
8. Scenario Walkthrough	9

1. Introduction

The project aims to mimic the web service provided by a business-to-consumer (B2C) e-commerce website. Whilst an e-commerce website covers an extensive range of functionalities, the team will be focussing on the process of a user making an online purchase. Hence, the scope of the project will revolve around a customer's journey from having an intention to purchase a product to making an order and finally tracking the order.

2. Technical Overview Diagram



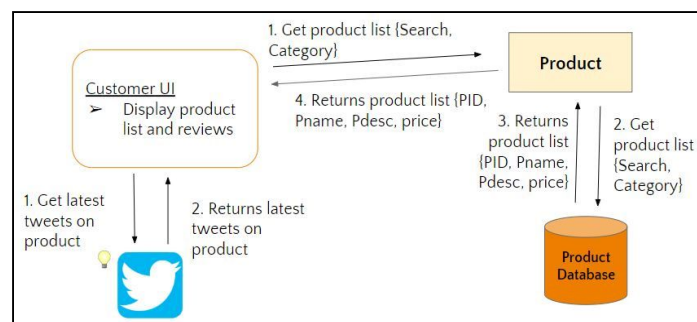
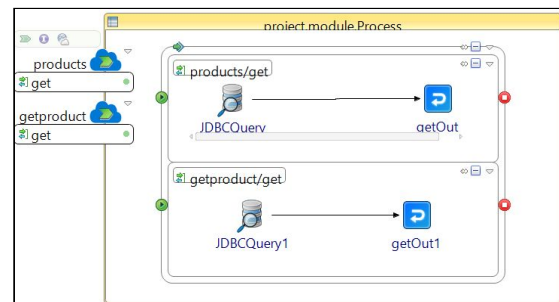
3. User Scenarios

3.1 User Scenario 1

3.1.1 User searches for products to view descriptions

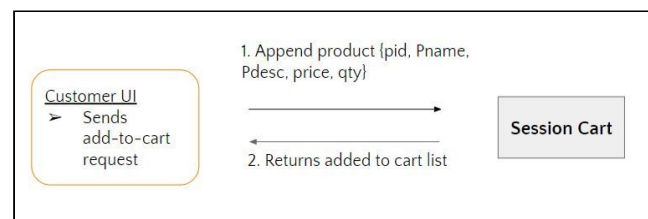
The following user scenario involves the activity of retrieving the lists of products available after the user enters the search criteria on the homepage of the UI, with the parameters search and category. The UI invokes the product service via GET to retrieve the product with matching search requests based on its category and product name from the Product database. This operation performs a JDBC Query into the Product table in the project_product database. The SQL Query statement is as follows: “*SELECT * FROM product where Pname like ? and category like ?*” This service will then return a 200 status success code and the matching product(s) attributes (pid, pname, pdesc, price, availability, category) via the getOut activity.

Aside from the Product microservice, an external Twitter API has been used to retrieve the latest tweets about the products and display them on the UI. The product name is used as the query parameter, a GET request is sent to the Twitter API using the TwitterAPIExchange library, then returning the result on the UI.



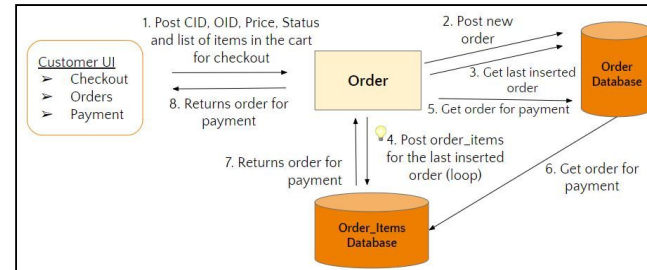
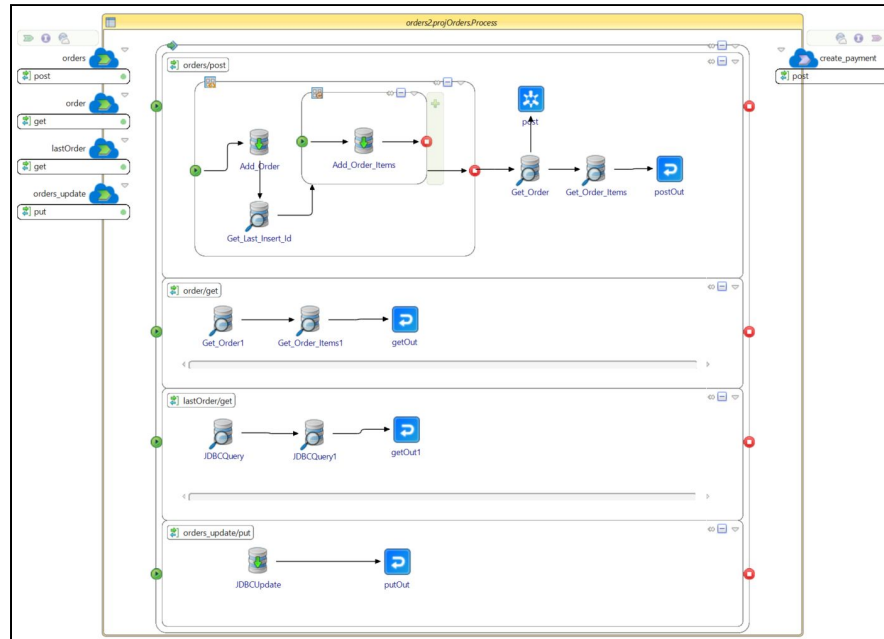
3.1.2 User adds product(s) to cart

Upon clicking on the “Add to cart” button on the UI, a cart session is created using PHP. Similarly on page load, a JQuery function will call the Get Product operation to perform a JDBC Query on the Product table of the project_product database with a SQL statement “*SELECT * FROM product WHERE PID = ?*”. It takes in an input value of the product's PID and returns a 200 status success code together with the product attributes (pid, pname, pdesc, price, availability, category) via the getOut1 activity. The product attributes are inserted into the cart session using PHP before sending back to the JQuery function on success. Lastly, the total price in a cart is also calculated here using PHP.



3.2 User Scenario 2

3.2.1 User checks out order



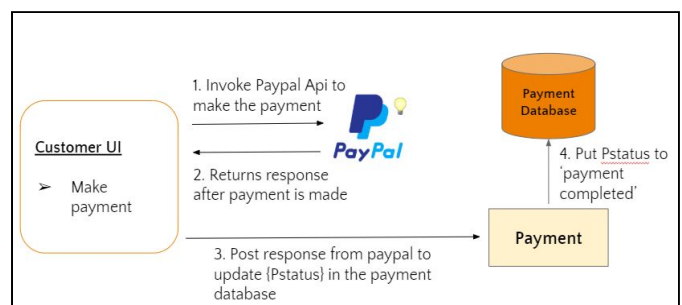
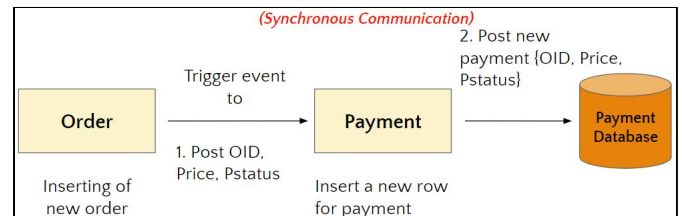
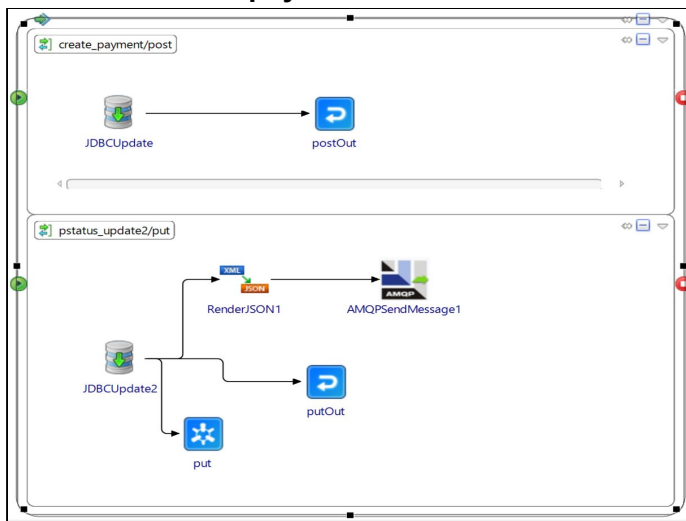
In this scenario, upon clicking on the checkout button on the UI, the UI would invoke the order service via AJAX POST, adding an order to the Add New Order operation, passing the customer name (CID), totalPrice, Pstatus and a list of order items to the operation.

Firstly, a JDBC insert, inserting the CID, totalPrice and Pstatus into the order database, corder table via a JDBC Update activity. The service then proceeds to get the last inserted OID via a JDBC Query activity. This is done by

a “SELECT LAST_INSERT_ID(” SQL statement in the activity. After getting the last inserted OID, the service will add the list of order items, consisting of PID, Pname, price, quantity and the last inserted OID attached to the item, taken from the previous activity, into the order_items table under the Order database.

Thereafter, the service will get the order and the list of order items of the last inserted OID from the database via JDBC Query activity. This service then returns a 201 status and last inserted order, together with its list of order items. This response that we get from this service will be used to display the list of order items the user is purchasing before they proceed to make the payment. A command synchronous communication between the Order and Payment microservices is used to trigger the Post New Payment operation in the Payment service.

3.2.2 User makes payment



Upon creating a new order in the corder database, the command synchronous communication with Payment service triggers the Post New Payment operation. First, the Order service will invoke the Payment service via POST by inputting the OID, price, Pstatus (“Order Created”) to the Post New Payment operation. The operation does a JDBC Insert into the Payment database, Payment table through the JDBC Update activity. The service then returns a 200 status, along with the details (PayID, OID, price, Pstatus) of the newly inserted payment.

Then, the user would make the payment via the customer UI which would invoke the PayPal API. Upon the completion of the payment in PayPal, PayPal would return the success response via the customer UI. The customer UI will then invoke the Payment service via PUT to the Update Payment operation by passing the payment status returned by the PayPal API to the Payment Service. This operation does a JDBC update of the Pstatus for the last inserted payment in the Payment table from the Payment database via the JDBC Update activity. The SQL statement specified in the JDBC Update activity is as follows, “UPDATE payment SET Pstatus = ? ORDER BY payid DESC LIMIT 1”.

After the Pstatus has been updated in the payment table, it will trigger a command synchronous communication with the Order service. This would invoke the order service via PUT to the Update Status operation by passing the pstatus (“Payment Completed”) from payment table to the order service. The Update Status operation will then do a JDBC Update of the Pstatus in the Order table from the order database, to “Payment Completed” for the latest order. This would later be used in the Notifications service.

Together with the command synchronous communication with the Order service, there will also be an event asynchronous communication with the Notifications service which will be explained in the next scenario.

3.3 User Scenario 3

3.3.1 User receives payment confirmation email upon successful payment

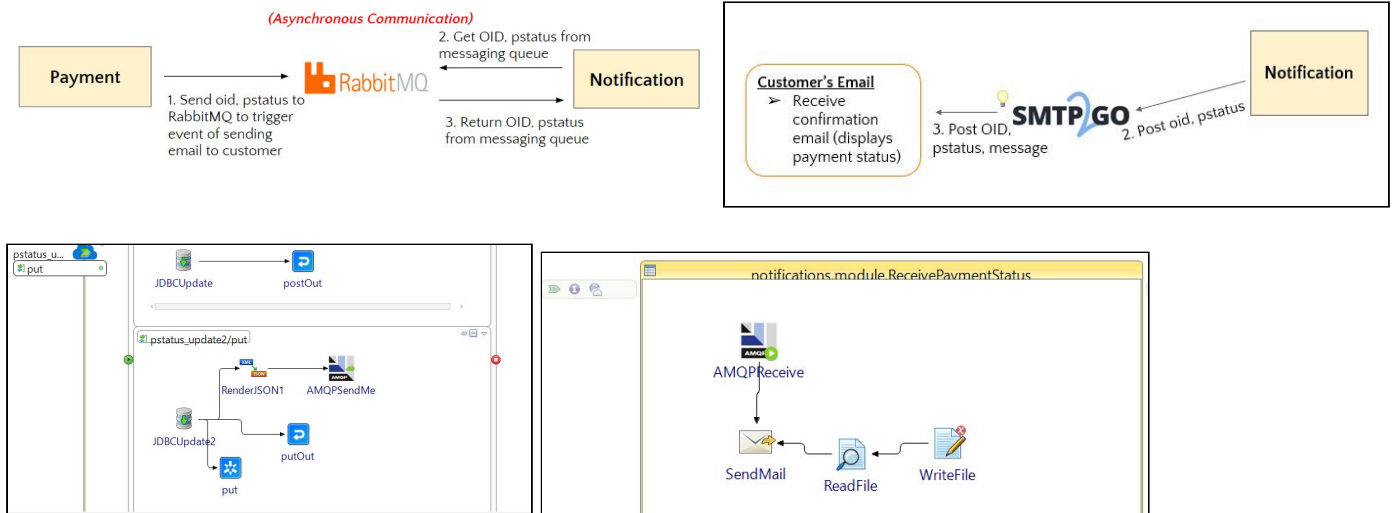


Figure 1: Payment service

Figure 2: Notifications service

Upon updating the Pstatus in the Payment database, an event asynchronous communication between the Payment service and Notifications service is triggered. A RenderJSON activity is used to retrieve the XML data of payment on the Payment database. It converts the XML data into a JSON string which is then passed to the AMQPSendMe activity. This activity sends the Pstatus = “payment completed” and OID (in JSON string format) to RabbitMQ by adding a queue. The service returns a 200 status response code with the updated Pstatus = “payment completed” via the putOut activity.

Next, the notifications microservice uses the AMQPReceive activity to retrieve the order (OID, Pstatus) asynchronously from the messaging queue in RabbitMQ. An external SMTP2GO API which has already been integrated into TIBCO is then used to send emails to the user. The SendMail activity retrieves the OID and Pstatus from the AMQPReceive Activity output and ReadFile that reads the default message which was written for the customer in Writefile, and includes these information in the email notification message to the user. Upon successful execution of the activity, the Notifications service will return a 200 status code.

3.3.2 User tracks order

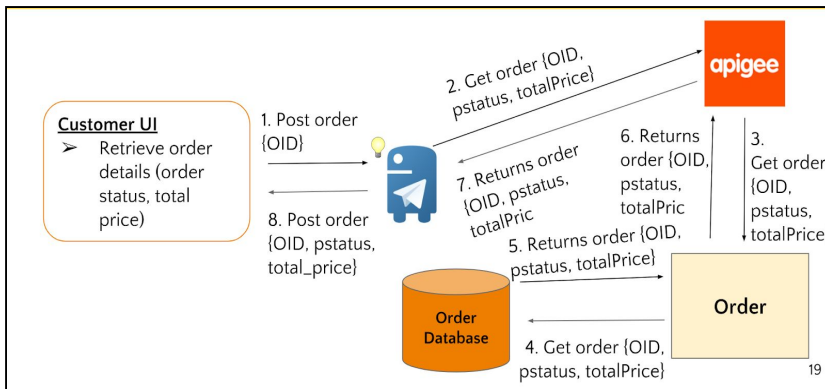


Figure 3: Order service

To track the previous orders, this scenario illustrates how the user can do so via Telegram. He can start the service by typing /start which the Telebot will request the user to input /getorders. This will then prompt the user to enter the OID to check for the status of the order requested. After inputting the OID desired, the Telebot will process the request via the webhook that was configured. The webhook established is used to process the inputs of the user to return the order status with the parameters {OID, Pstatus, totalPrice} by invoking the Get Order operation from the Order service which is held on localhost. To ensure the API can be accessed online, we used Apigee to do a reverse proxy with ngrok to establish a forwarding URL to tunnel to localhost:80 to handle the request. As such, Apigee gets the order with parameters {OID, Pstatus, totalPrice} from the Order service in the Order table in the Order database with the SQL statement: *SELECT * FROM*

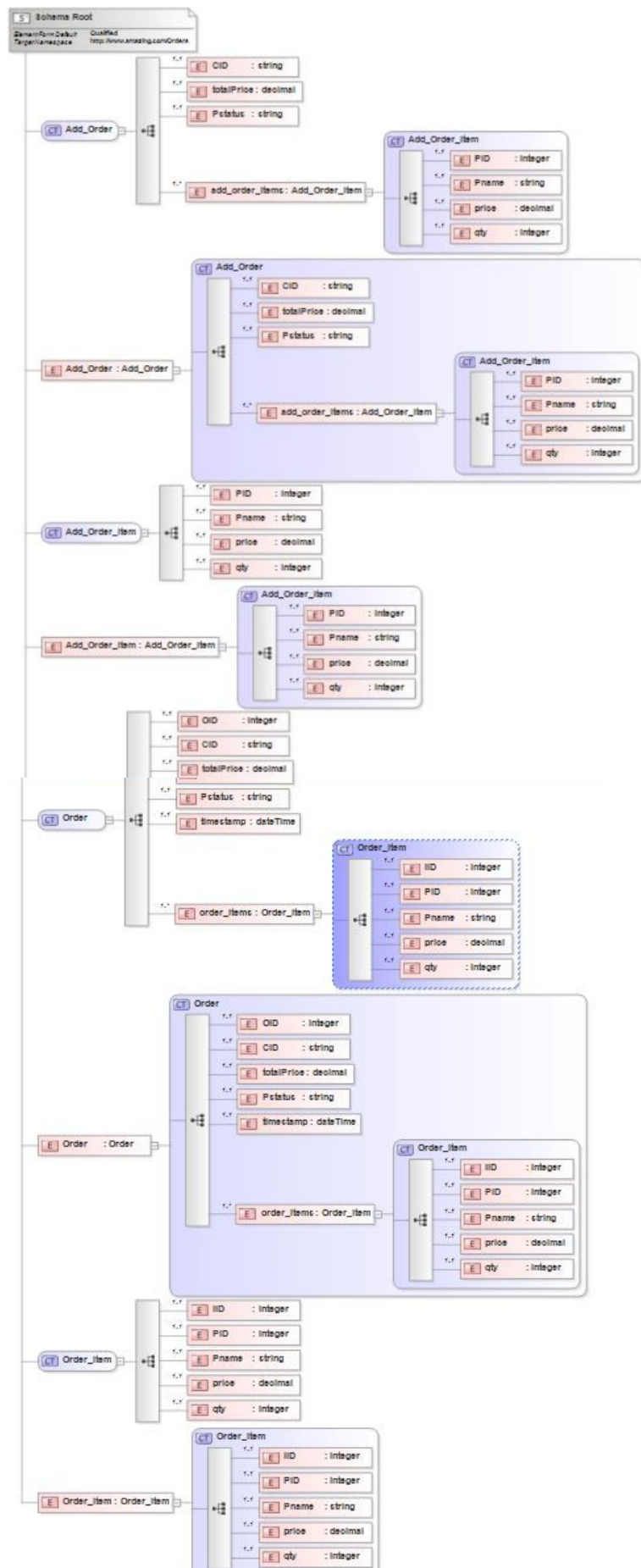
CORDER WHERE OID = ?. Subsequently this returns the order details of that specific OID with the parameters {OID, Pstatus, totalPrice} back to the Order service, Apigee and lastly the Telebot. The Telebot will then display the OID, Pstatus and totalPrice on the user's telegram.

4. Web Services

Service	Operation	Description	Input	Output
Product service	Get all products	GET /products/{Pname}&{category}	Pname, category	array of products
	Get product	GET /getproduct/{PID}	PID	product
Order service	Post new order	POST /orders	Array of products	Nil
	Get last order	GET /lastOrder	Nil	Array of order details
	Put payment status update	PUT /orders_update	Pstatus	Nil
	Get order	GET /order/{OID}	OID	Array of order details
Payment service	Post new payment	POST /create_payment	OID, Pstatus, price	Nil
	Put payment status update	PUT /pstatus_update2	Pstatus	Nil
Notifications service	Post OID and Pstatus for messaging	Post oid and pstatus to SMTP2GO API (integrated on TIBCO), where parameters received via GET OID, Pstatus from RabbitMQ messaging queue	OID, Pstatus, Readfile text	SendMail merged text content
Twitter API	N/A	https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets GET	Pname	Latest tweet
Paypal API	N/A	Post payment details (payment status) upon creation of new order (i.e. payment) via PayPal https://developer.paypal.com/docs/api/orders/v2/ POST /v2/checkout/orders GET /v2/checkout/orders/{order_id}	[POST] totalPrice Access-token [GET] order_id Access-token	Status
Telebot API	N/A	https://core.telegram.org/bots/api GET	OID	Array of order details

5. Usage of XML schema

This schema diagram was used for our customer's order scenario in User Scenarios **3.2.1 - 3.2.2**. It is considered the most interesting XML schema out of the 5 schemas we have as it contains the most number of complex types and elements. On top of this, we have interesting parent element `order_items` which holds different elements of an item.



6. Graphical User Interface

With Web-based application, our group utilised different programming languages including HTML, CSS, PHP, JS (JQUERY, JSON, AJAX). Subsequently, we also have the user email UI and lastly, Telegram for tracking order status.

7. Going Beyond the Labs

7.1 Twitter API

The Twitter API is used to retrieve the most recent tweets from Twitter regarding the product that we are selling on our ecommerce webpage. We did this by utilizing the TwitterAPIExchange library that is written in PHP to help us call the Twitter API functions easily. The input parameter for our Twitter API URL would be the product name which will get us the latest tweets. We then use AJAX to process the information retrieved from the Twitter API to be displayed on our ecommerce website.

7.2 TIBCO For-Loop

The loop function allows the service to run again and again until it reaches a stop point. This is necessary to insert or populate the order_items into a specific order. When the database tables are normalised, this is essential as order and order_items tables has a one to many relationship, hence, one order can have many order_items. In order to sift out all the items, we have to loop the order.

7.3 PayPal API

An external PayPal order API has been integrated into the project to create a new payment when the customer checks out his list of order items. The user will click on a button (provided by PayPal) which allows him to log in to his PayPal account and make a payment on PayPal. Likewise on the backend server, after the payment is completed on PayPal, the order_id in the JSON response will be stored in a cookie, thus making it easily accessed in one single session. The order_id cookie will be placed into the PayPal GET API, returning a JSON response that includes payment status. This API is called through the use of cURL, on the PHP backend server. After retrieving the relevant order's details, the Pstatus is updated in the Payment database by invoking the AJAX PUT on the Payment microservice.

7.4 Email

SMTP2Go is used to provide reliable and scalable email delivery service to notify our customers on their successful verification of payment made through the PayPal API. SMTP2Go is implemented in conjunction with SendMail, Readfile and Writefile activity provided in TIBCO. RabbitMQ is used as an asynchronous messaging protocol to store OID and Pstatus from Payment microservice which will then be received by Notifications microservice to invoke this Email service. This output from RabbitMQ {OID, Pstatus} will then be included together with a default message written in Writefile and subsequently send it to the purchaser's email.

7.5 Telegram Bot

Apart from the Email service, Telegram bot is used to notify our customers on their order status and total price by having the customer to chat with the Telegram bot with OID as their input. Telegram bot API is integrated into the Telegram bot to authenticate a valid user with its unique token and process requests over a webhook that has a HTTPS URL. We used Apigee as it has the HTTPS URL capabilities so as to tunnel into the localhost:80 to handle the request. In our project, Telegram bot retrieves user's first name upon typing */start* and post a *Hi with his/her first name*. It then prompts the user to type */getorders* to get order and the order ID to check order details for a specific order ID. With an in-depth explanation on how the order detail is retrieved from the scenario write up, the Telegram bot displays the information for the customer.

8. Scenario Walkthrough

1. Customer accesses shop's webpage and searches for the product that he is looking for.

Buy First, Think Later!

Search

Category

All

Submit →

2. Customer views all products description and Twitter users' reviews listed on web by clicking 'Submit' button, with category set to 'All'.




Buy First, Think Later!

Search

Category

All

Submit →

Image	Product	Category	Price	Add to Cart
	Sekiro Description: One Handed Samurai goes on a mission to save prince Latest tweet: RT @pcgamer: I beat Sekiro's final boss with cheats and I feel fine https://t.co/Fj4i8d6sUb https://t.co/N38RL5zxZO	games	69.99	Add To Cart
	Air Max 1 Description: Cool Shoes Latest tweet: Nike Air Max winter shoe 95 royal blue https://t.co/nlapUf0d7n	shoes	249.99	Add To Cart
	Iphone X Description: Steve Jobs is disappointed	phones	1249.99	Add To Cart

3. Upon clicking 'Add to cart' in Step #2, user is directed to cart.php, which displays the list of products in cart

Shopping Cart

No.	Title	Price	Quantity	Amount
1	Sekiro	\$69.99	1	\$69.99
Total				\$69.99

My name:

Checkout

Continue Shopping

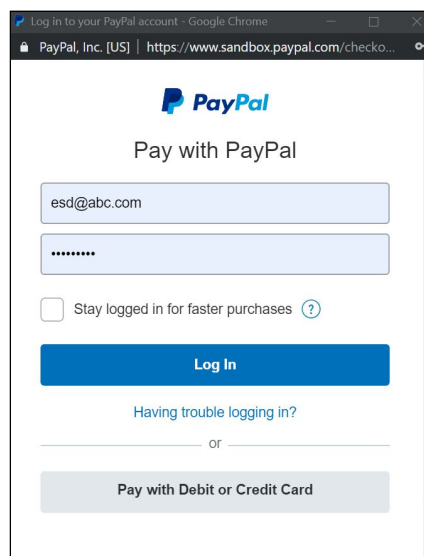
4. User clicks on **'Checkout'** in Step #3, and is redirected to payment.php page.

Order Confirmation

No.	Title	Quantity	Price	Amount
1	Sekiro	1	\$ 69.99	\$ 69.99
Total				\$ 69.99

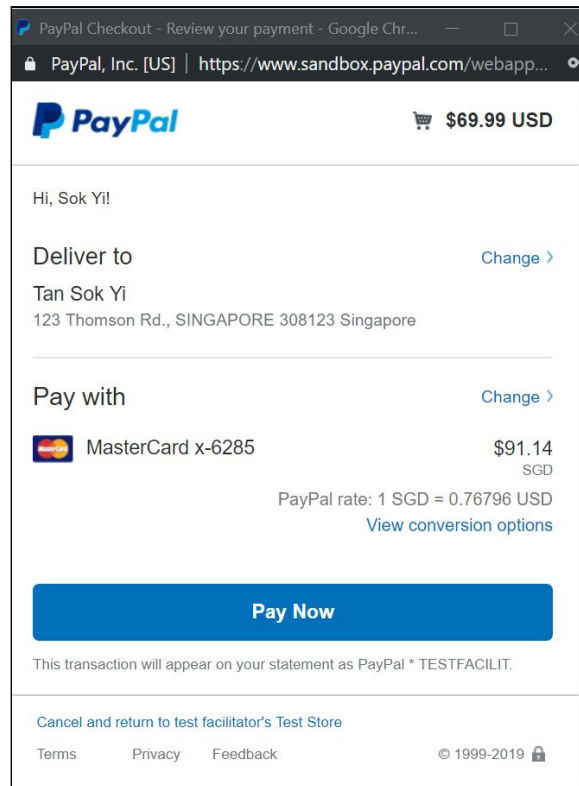


5. Assuming that the user owns a PayPal account, the user will be making a payment by clicking on the **'PayPal'** button. The PayPal login window pops up.



The screenshot shows a web browser window titled "Log in to your PayPal account - Google Chrome". The address bar shows "PayPal, Inc. [US] | https://www.sandbox.paypal.com/checko...". The main content area displays the PayPal logo and the text "Pay with PayPal". Below this, there is a login form with two input fields: the first contains the email address "esd@abc.com" and the second contains masked characters "*****". A checkbox labeled "Stay logged in for faster purchases" with a question mark icon is positioned below the password field. A prominent blue "Log In" button is centered below the checkbox. Underneath the button, there is a link that says "Having trouble logging in?". At the bottom of the form, there is a light gray button labeled "Pay with Debit or Credit Card".

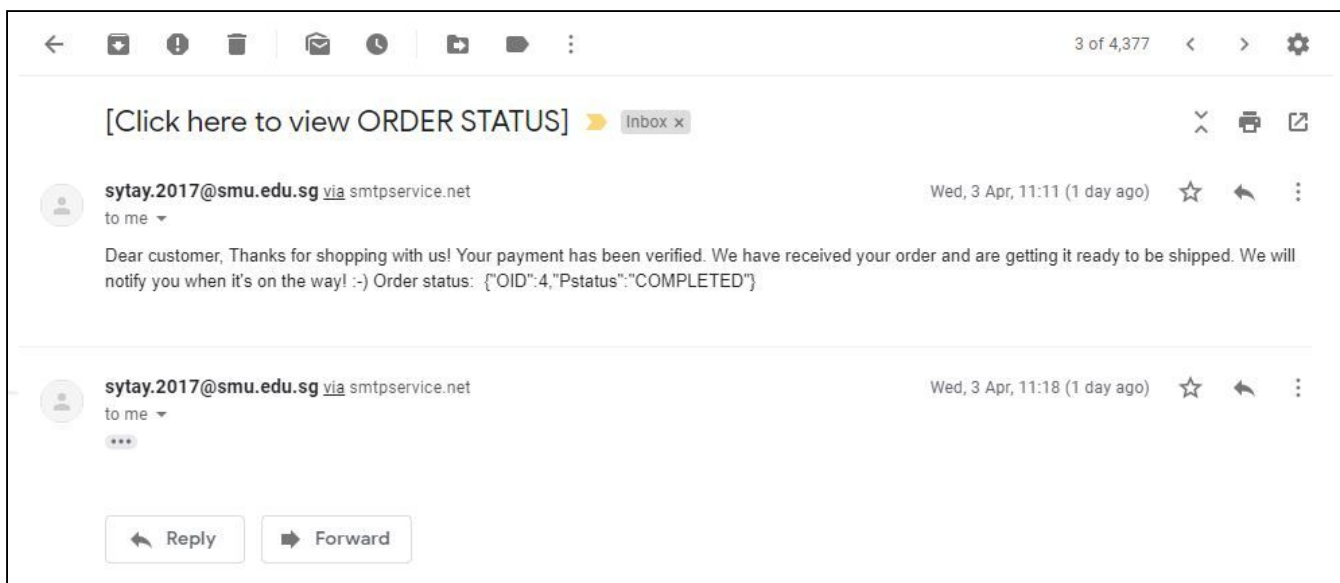
6. After logging in, the user confirms the payment details and proceeds to click on the 'Pay Now' button.



7. User is redirected to payment_completion.php page, after completing the payment.

Your order is on its way!

8. Upon completion of payment, the user receives a payment confirmation email.



9. After confirming payment, user goes on to check order status by using a Telegram bot, @g6t4_bot .

