|  |
| --- |
|  |

|  |
| --- |
| **IS213 Enterprise Solution Development | *G7T3***  **Assignment Report**  *Gracia Yuwono Kwantalalu*  *Hamizan Ali Bin Azmee*  *Madhumitha D/O Suresh Kumar*  *R Jeyavani*  *Shurti D/O Thirunaukrarasu*  *Tay Jie Wen Marcus* |

[**Introduction**](#_ojwf8dabkrxt) **3**

[**Business & User Scenario 1 - Customer Adds Product to Cart & Views Cart**](#_tyjcwt) **3**

[**Business & User Scenario 2 - Customer Payment via Paypal**](#_3dy6vkm) **3**

[**Business & User Scenario 3 - Order Dispatch**](#_1t3h5sf) **3**

[**Technical Overview Diagram**](#_9bqv3xk1tt6h) **3**

[**Scenario 1 - Customer Adds Product to Cart & Views Cart**](#_7hujjxavyqo0) **4**

[**User Scenario 2 - Customer Payment via Paypal**](#_v6xgd6i50tk3) **4**

[**Scenario 3 - Order Dispatch**](#_m2ebx2u41jk8) **6**

[**Web Services**](#_2s8eyo1) **7**

[**Usage of XML schema**](#_17dp8vu) **8**

[**Graphical User Interface**](#_3rdcrjn) **8**

[**Beyond the Labs (Scenarios 2 & 3) - Invoking PayPal APIs & Sending Email**](#_lnxbz9) **9**

# **Introduction**

Seyfora is a cosmetic company with an e-commerce platform. Our 3 key scenarios include customer adding products to cart, customer making payment for the items ordered and customer being sent an email notification about order shipment.

## **Business & User Scenario 1 - Customer Adds Product to Cart & Views Cart**

1. View product list: When the customer goes to Seyfora’s home website, she is able to see a list of products which Seyfora sells. 2.Add items to cart: Customer may then add the items she wants to purchase into the cart. 3.View items added to cart: After the customer adds all the products she wants to purchase, she can click on the ‘view cart’ button to see all the items she has added into her cart.

## **Business & User Scenario 2 - Customer Payment via Paypal**

1. View loyalty points: Customer is able to see how many points she has. One point is equivalent to one dollar. In this scenario, the customer is assumed to use all the loyalty points she has. 2. Make payment: Customer will be redirected to the paypal website to execute payment.

## **Business & User Scenario 3 - Order Dispatch**

This scenario represents the business situation of an item being sent for shipping. Upon a staff printing a shipping label and affixing it on the packaging, the details of which are then user-updated into the enterprise system. The scenario ends with the enterprise system sending an email to the shipping recipient and the UI reloading upon successful scenario execution.

# **Technical Overview Diagram**

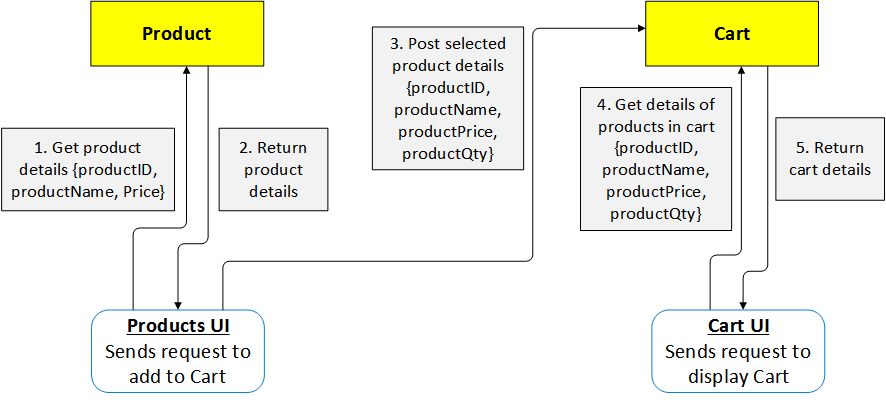
A screenshot of a cell phone

Description automatically generated

**User Scenarios**

## **Scenario 1 *-* Customer Adds Product to Cart & Views Cart**

*User Scenario Diagram*

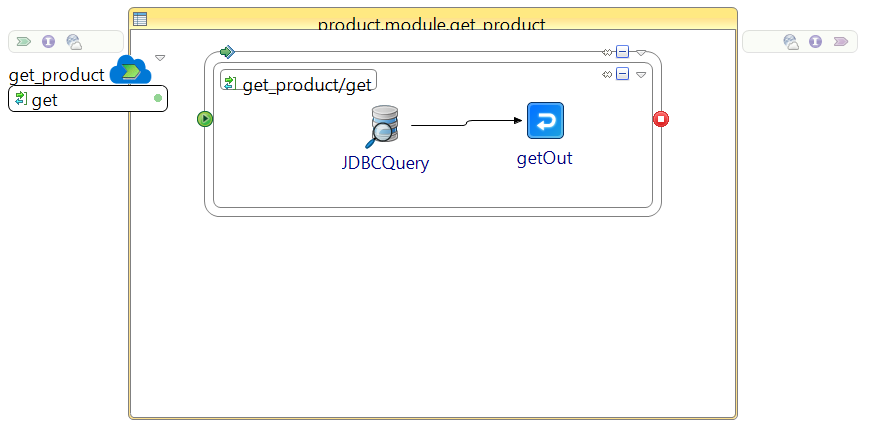
**

*Process Definition Diagrams*

After customer is logged in (assumed that she is), a cart with a unique ID is created for her. The customer is able to see a **list of Seyfora products in the Product UI via a GET operation**. Product information including the ID, name and price are retrieved from the product database and can be seen by the customer. Customer then inputs the quantities for each items she wishes to purchase. When the user clicks on the ‘add to cart’ button, this invokes the cart service to carry out a **POST operation to insert the details of the selected items (productID, productName, productPrice, productQty)** into the cart\_items table.

She can then click on ‘view cart’ to see the items she has added. This will redirect her to the cart UI page. In the backend, **total price and points of the cart table is updated via a PUT operation** to allow for payment later on. The customer is able to see the items in her cart, loyalty points which is obtained through a **GET call of the customer service**, and total amount payable.

## 



*Product values are retrieved for customer to view on UI*

## 

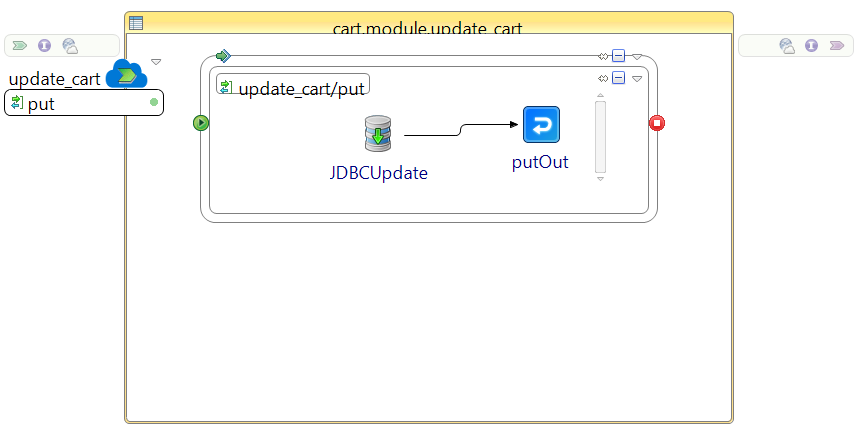
*An empty cart is created*

## 

## 

*Get the loyalty point of a particular customer (hardcoded customer ID)*

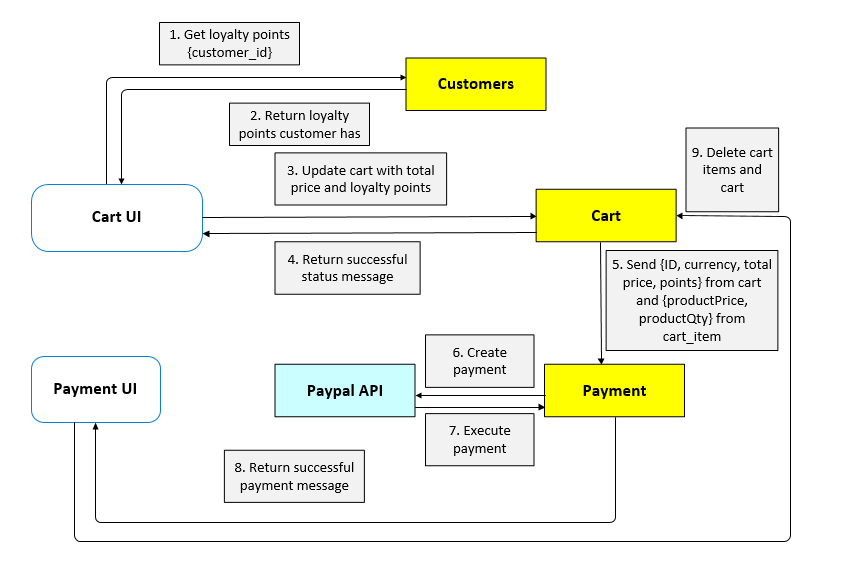
*Items selected by the customer is inserted into the cart\_items table*



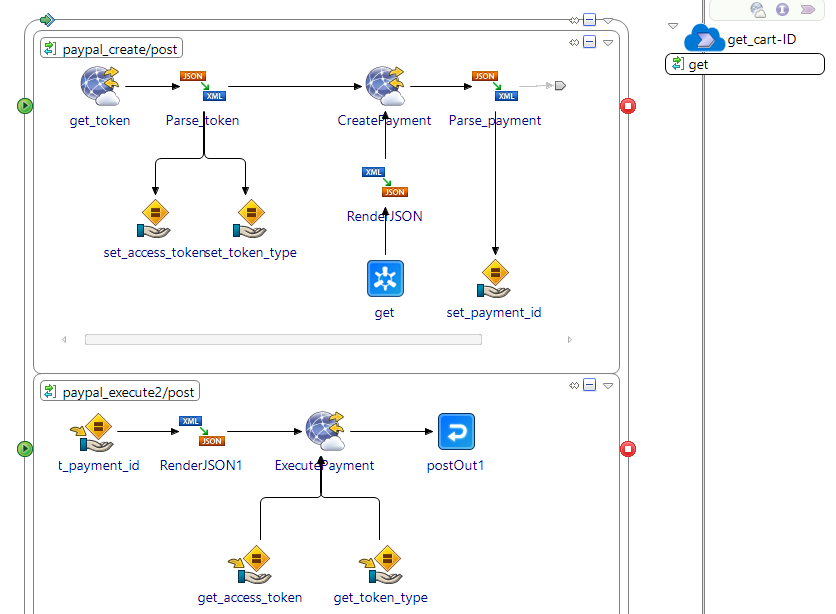
*‘Total’ and ‘points’ in cart table are updated with the calculated total price and loyalty points of the user*

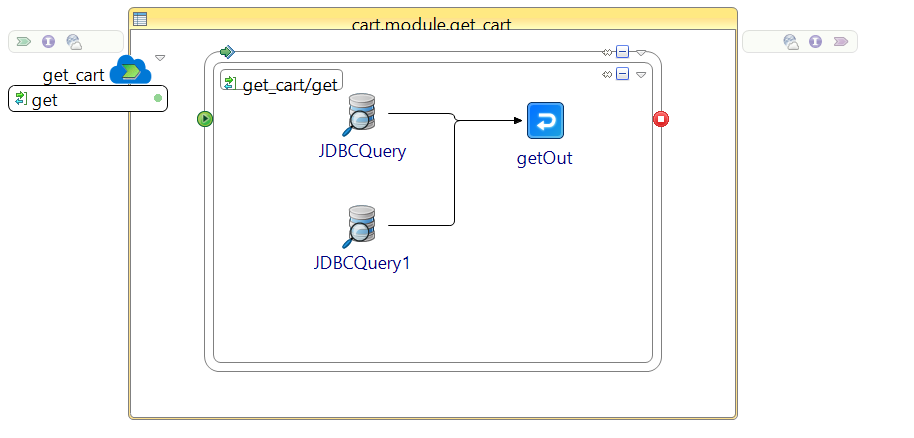
## **Scenario 2 *-* Customer Payment via Paypal**

*User Scenario Diagram*



*Process Definition Diagram*



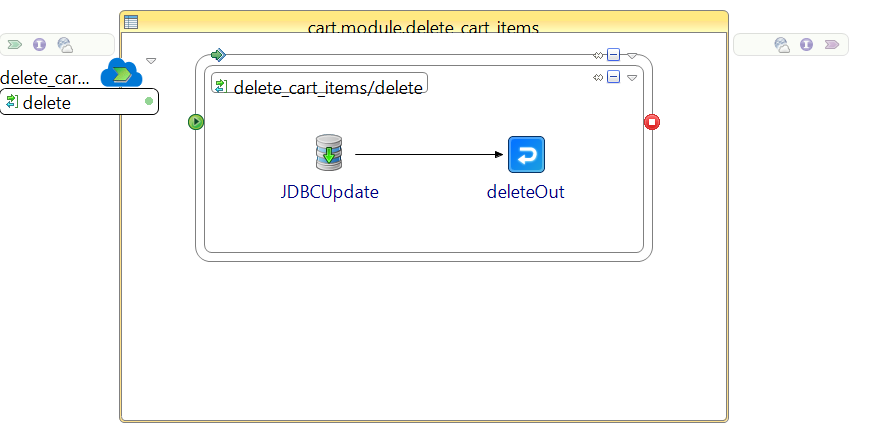
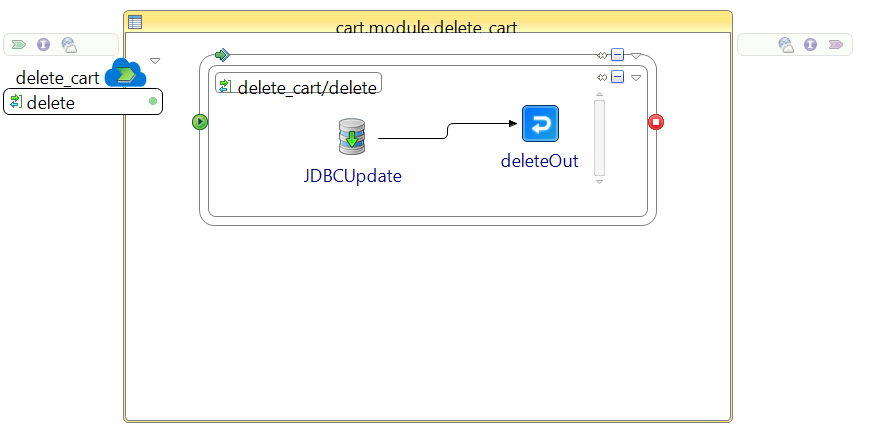


*delete\_cart invoked to empty cart once payment has been made*

*Payment invoked to carry out payment by PayPal API*

*delete\_cart\_items invoked to empty cart\_items once payment has been made*

*cart and cart\_items invoked to create payment*

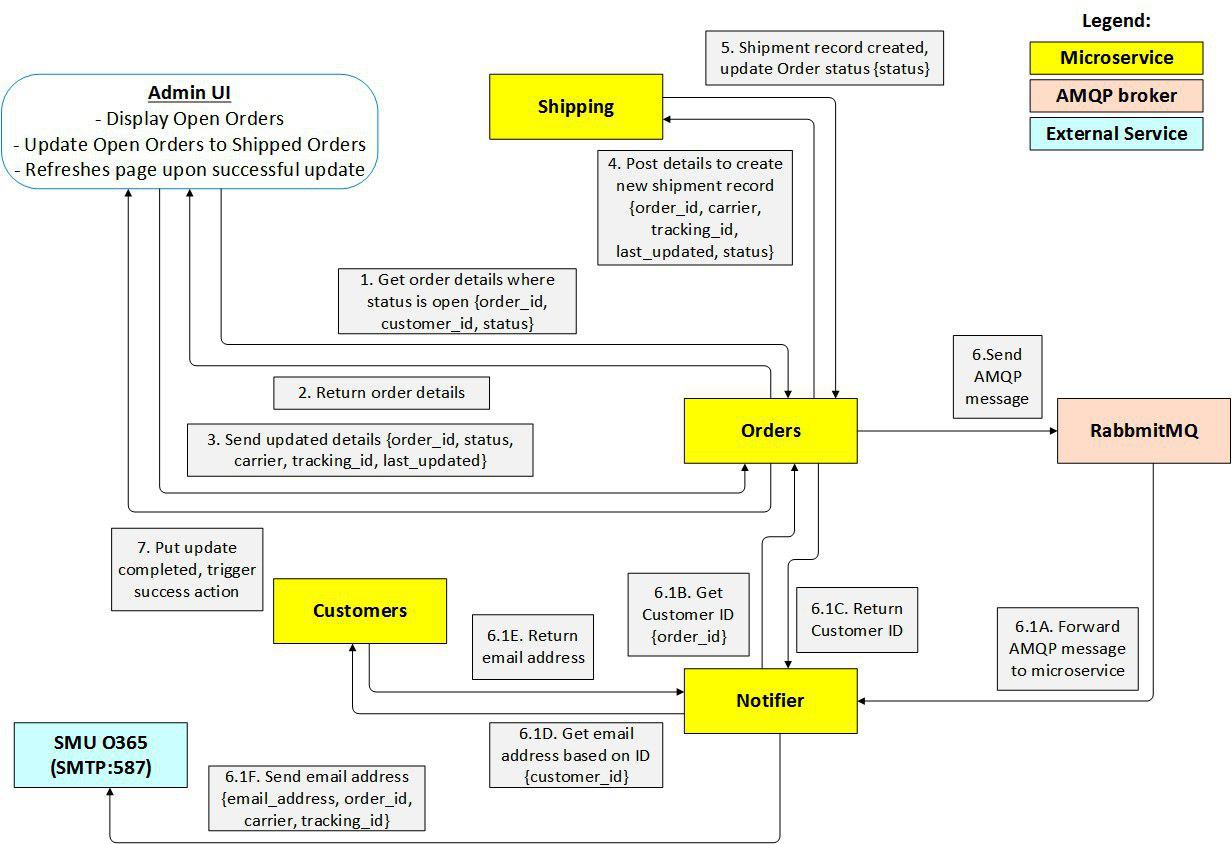


Customer can click on the ‘proceed to payment’ button to pay for the items. The customer is then redirected to paypal’s website. The PayPal API **invokes the cart service via a query, GET operation (by referencing to the cart service in tibco)** Information including the ID, total price, currency, points, product price and quantity will be obtained to create payment on the API.

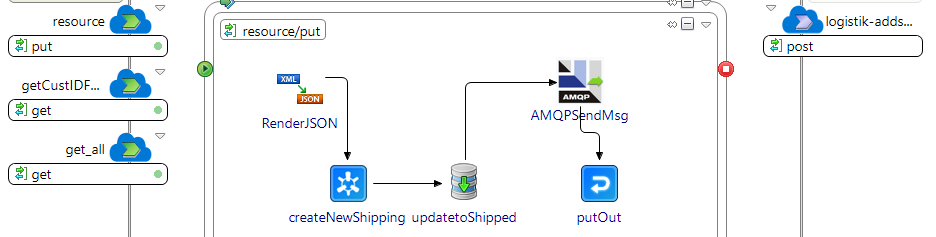
Simultaneously, an access token is taken from PayPal using the Create Payment service to authenticate the customer. Together with this information, the payment ID and payer ID are obtained as they are needed as inputs for the execute payment service. The payment ID is obtained from the create payment service while we get the payer ID from the UI. After payment is executed successfully, customer will be redirected to the home page after seeing a success message. In the backend, the cart\_items and cart for that customer are deleted so that it can be initialised for the next use.

## **Scenario 3 *-* Order Dispatch**

*User Scenario Diagram*

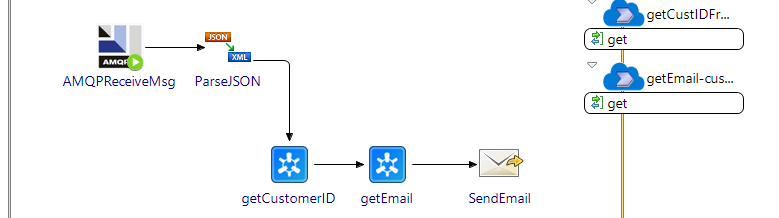
**

Process Definition Diagram(s)



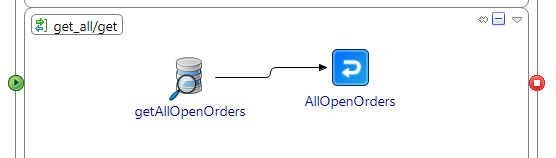
*Process which populates the UI*

*Admin\_Order process (‘resource’) called in Admin UI, calls Notifier microservice asynchronously*



*Notifier microservice. An event-based process triggered triggered via AMQP*

*Gets all Order records with “Open” status*



The UI is initialised by calling a query-based get\_all service function under Admin\_Order microservice. The UI will pre-populate a HTML table with existing information, with the staff now required to input the Carrier and Tracking ID information. Upon a staff clicking “Send Order” with completed details in the UI, it triggers the main put service (“resource”) function under Admin\_Order microservice. The command function first references Logistik microservice to create a new shipping record in the shipping table of Logistik database. Following successful execution, it then updates the Orders table, changing the Status column from “Open” to “Shipped”. After successful database update, the main process continues to send an AMQP message to the Notifications queue which will then be received by the Notifier microservice. The Admin\_Order resource process then ends.

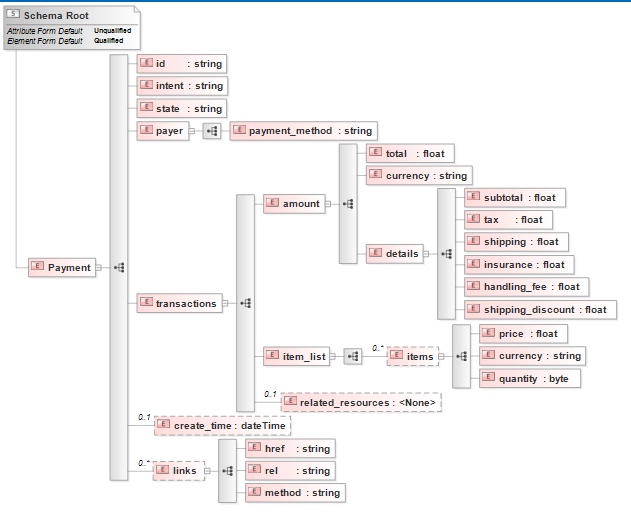
The receipt of AMQP (via RabbitMQ broker) triggers the event-based Notifier service. It utilises part of the forwarded data (order\_id) and queries upon Admin\_Order microservice to obtain the corresponding customerID belonging to the specific Order being shipped. With the customerID now known, Notifier proceeds to query via Customers service to obtain the specific email address and name of the customer. The final action in Notifier sends an email to the customer with existing information involved in the scenario process thus far (i.e. customer name, carrier, order ID, tracking ID).

# **Web Services**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Service | Operation | Description | Input | Output |
| *Product Service* | */get\_product*  *(GET)* | *Obtains a list of products to be displayed to the user on the product UI.* | *nil* | *Array of products (Product\_ID, productName, productPrice, productAvailability)* |
|
| *Cart Service* | */get\_cart/{ID}*  *(GET)* | *Get details of items that have been added into cart by the customer.* | *ID* | *Array of products (Product\_ID, productName, productPrice, productQty)* |
| */insert\_cart*  *(POST)* | *Adds in a row of ID, Total, Currency, Points. Total and Points have default values of 0 which will be updated later before making payment.* | *JSON Array of cart with fixed base values* | *Response status of 201 to show a successful add.* |
| */insert\_cart\_items*  *(POST)* | *Adds in a row of ID, productID, productName, productPrice, productQty* | *JSON Array of cart\_item with ID, productID, productName, productPrice, productQty values* | *Response status of 201 to show a successful add.* |
| */update\_cart*  *(PUT)* | *Updates Total price and Points of cart with the total\_payable variable from the UI and the customer loyalty points respectively* | *JSON Array of Cart with updated total\_price and points values* | *Response status of 202 to show a successful update.* |
| */delete\_cart*  *(DELETE)* | *Deletes the cart* | *nil* | *Response status of 200 to show a successful deletion.* |
| */delete\_cart\_items*  *(DELETE)* | *Deletes the items in a cart* | *nil* | *Response status of 200 to show a successful deletion.* |
| *Customer service* | */customer/{customer\_id}*  *(GET)* | *Get the loyalty point of a specific customer* | *Customer\_id* | *Customer object (customer\_id, customer\_name, points, birth\_date, contact\_number, email\_address)* |
| */getEmail/{customer\_id}* | *Get the customer email* | *Customer\_id* | *Email\_address* |
| *Payment Service* | */paypal\_create*  *(POST)* | *Once customer clicks on “proceed to payment” button on the payment UI, this service will create the payment so that the customer can make payment through paypal.* | *order\_id (cart\_id of the cart which is to be paid)* | *Array of links (href, rel, method)* |
| */paypal\_execute2*  *(POST)* | *After payment has been created, the customer is able to view the total payment cost and when the customer clicks on confirm payment, this service will execute the payment for the customer.* | *Payment\_id and payer\_id* | *JSON Array which can be seen in the console* |
| *Admin\_order service* | */shipOrder (PUT)* | *Upon clicking “Send Order”, the UI triggers the Admin\_Order service to command Logistik to create a shipment record. It also creates an event for Notifier to start the process of sending an email.* | *Values (order\_id, status, carrier, tracking\_id, last\_updated)* | *Response status of 200 to show a successful update.* |
| */get\_all (GET)* | *UI automatically calls this query function upon page load. Obtains all rows where the status is “Open”.* | *nil* | *JSON array (order\_id, customer\_id, status)* |
| *Office 365* | *N/A* | *Sends an email to a customer using a pre-configured account via SMU’s O365 SMTP.* | *[SendEmail-input] from, to, subject, bodyText* | *N/A (email received on client side)* |

# **Usage of XML schema**

Each microservice has its own XML Schema (XSD). This was to ensure standards and allow others to understand what the requirements for a specific microservice are. The Payment Schema was the most complex as seen below.



This schema has been developed to facilitate payment through the PayPal API. As PayPal has a very strict API documentation to follow, we have constructed this schema to take in required fields. After which, we use renderJSON to convert the XML text to JSON string and pass it into Create Payment REST API as one of the inputs so that the payment can be created, and relevant information can be displayed properly to the customer.

# **Graphical User Interface**

For our project, we used JQuery, Javascript and HTML to build the UI. We focused on the functionality rather than the aesthetics as this was something new to us. We chose to use jQuery instead of something familiar as it allows for a more interactive webpage where each action is validated with a response to let the user know that their input has been registered successfully. For example, when the customer adds a red lipstick by clicking the “add to cart” button, the user only receives the ‘item added to cart’ alert when the service is *successfully* invoked. Additionally, both pre-existing and newly-defined Javascript/JQuery functions (e.g. setTimeout, .fail) have also been used for error prevention and handling.

# **Beyond the Labs (Scenarios 2 & 3) - Invoking PayPal APIs & Sending Email**

**Create payment:** We had to find a way to get an access token from PayPal using a Sandbox account. After which, we still need to have the proper inputs from our get\_cart service using IPC to get information such as total price, currency, points, product price and product quantity for the payment to be created. We also needed to find the PayPal KeyStore (PPKeyStore) to ensure that we are authenticated to use the paypal service.

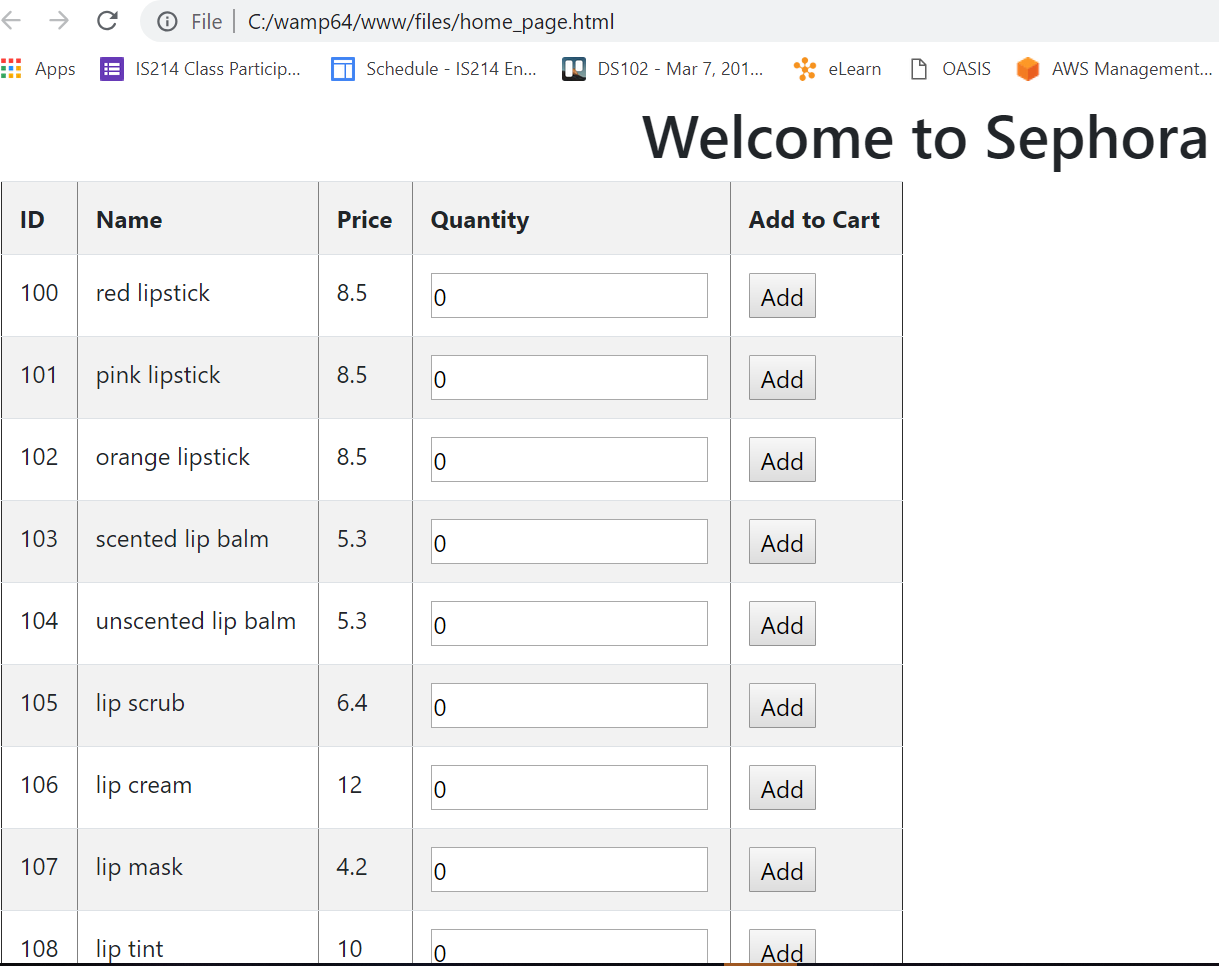
**Execute payment:** Executing payment only requires payment id and payer id. With these inputs, we are able to authorize the payment and execute them.

**Send Email:** To send an email simulating an internal corporate account, the Email palette in was utilised. Instead of directly connecting to an onprem email server, connection was made to SMU’s OWA via Office 365’s pre-configured port number with STARTTLS security settings. Using the existing organisational email account as the sender, customer’s email address as the recipient, we are able to send an email containing pertinent information using the process’s element values mapped to the bodyText via XPath concatenation.

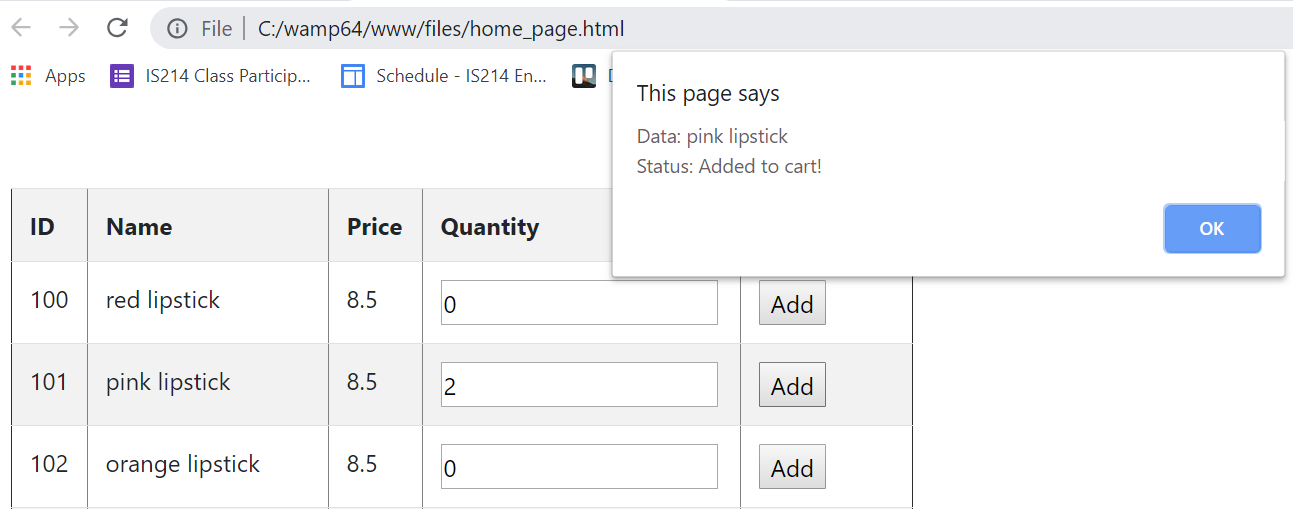
**Scenario 1-3 Walkthrough**

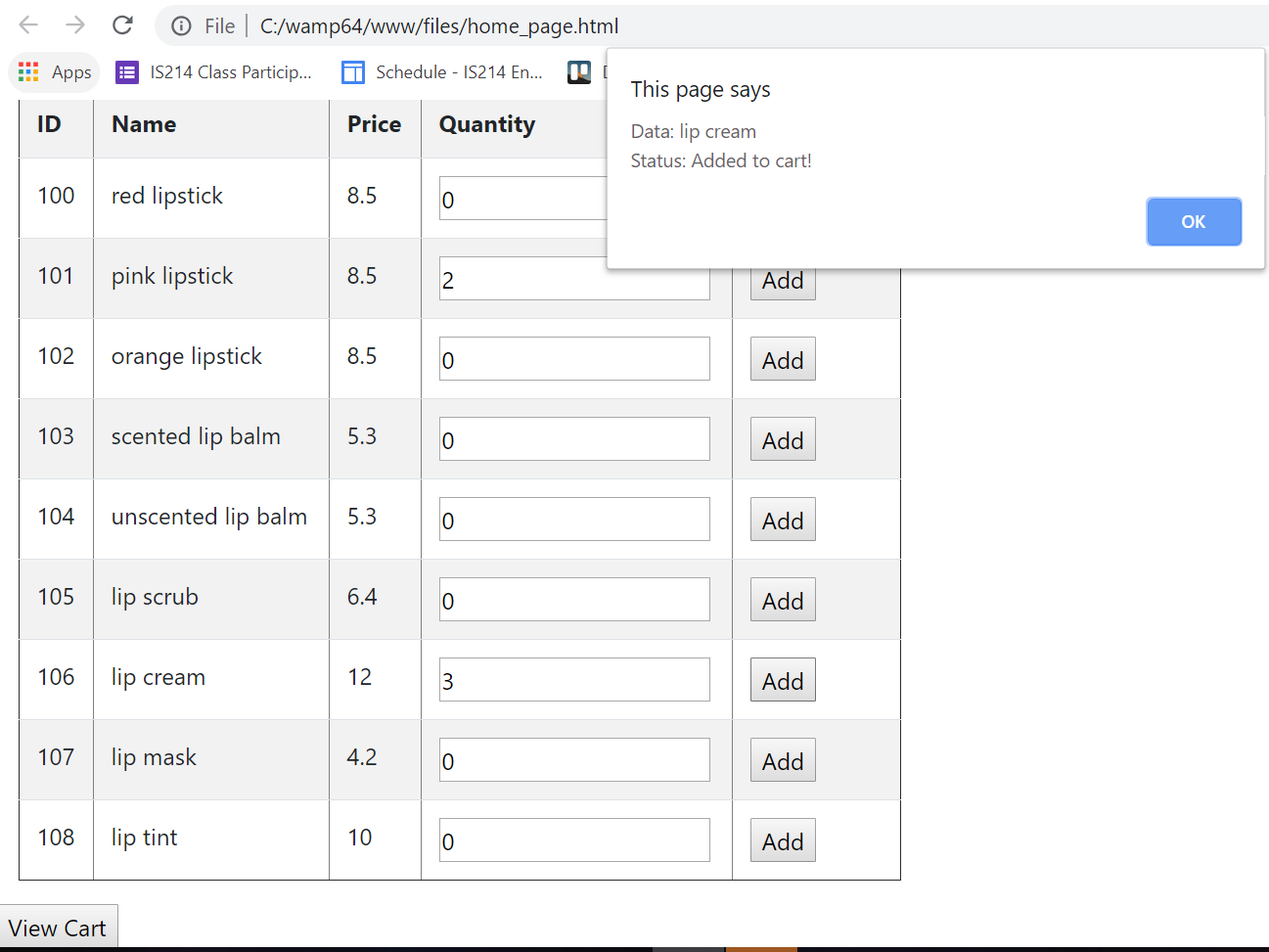
*Assume customer with customer\_id 12345678903 is already logged in. Cart ID 0 has been assigned to this customer. Navigate to http://<system name>/<designated Webroot folder>/homepage.html*

1. **[Scenario 1] Customer views homepage.html where products are displayed.**

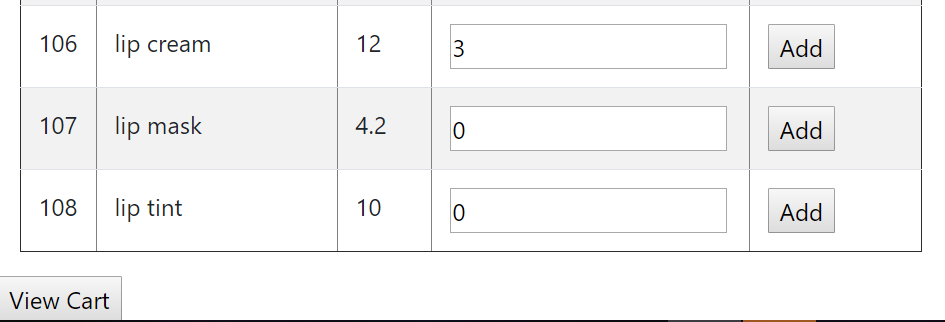


1. **For each item which the customer wants, she types in the desired quantity and presses the ‘Add’ button for that row. In this case, 2 pink lipsticks and 3 lip creams.**



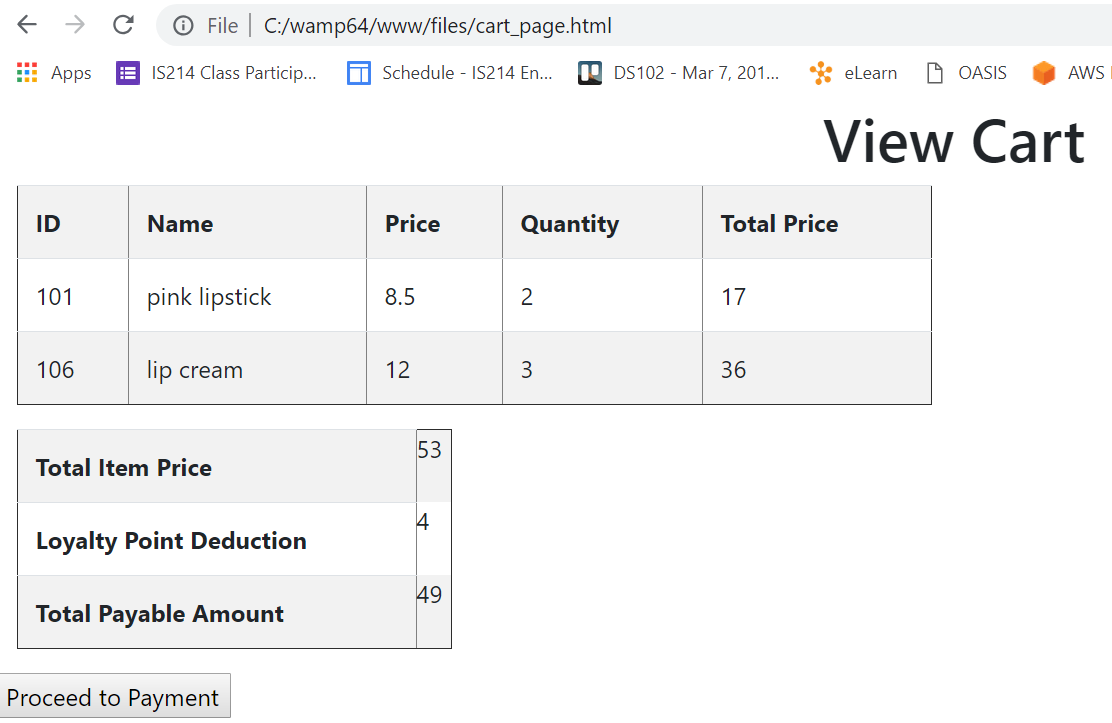


1. **After all desired items have been added, customer can now view their cart by clicking on the ‘View Cart’ button at the bottom of the page. (redirects to cart\_page.html)**

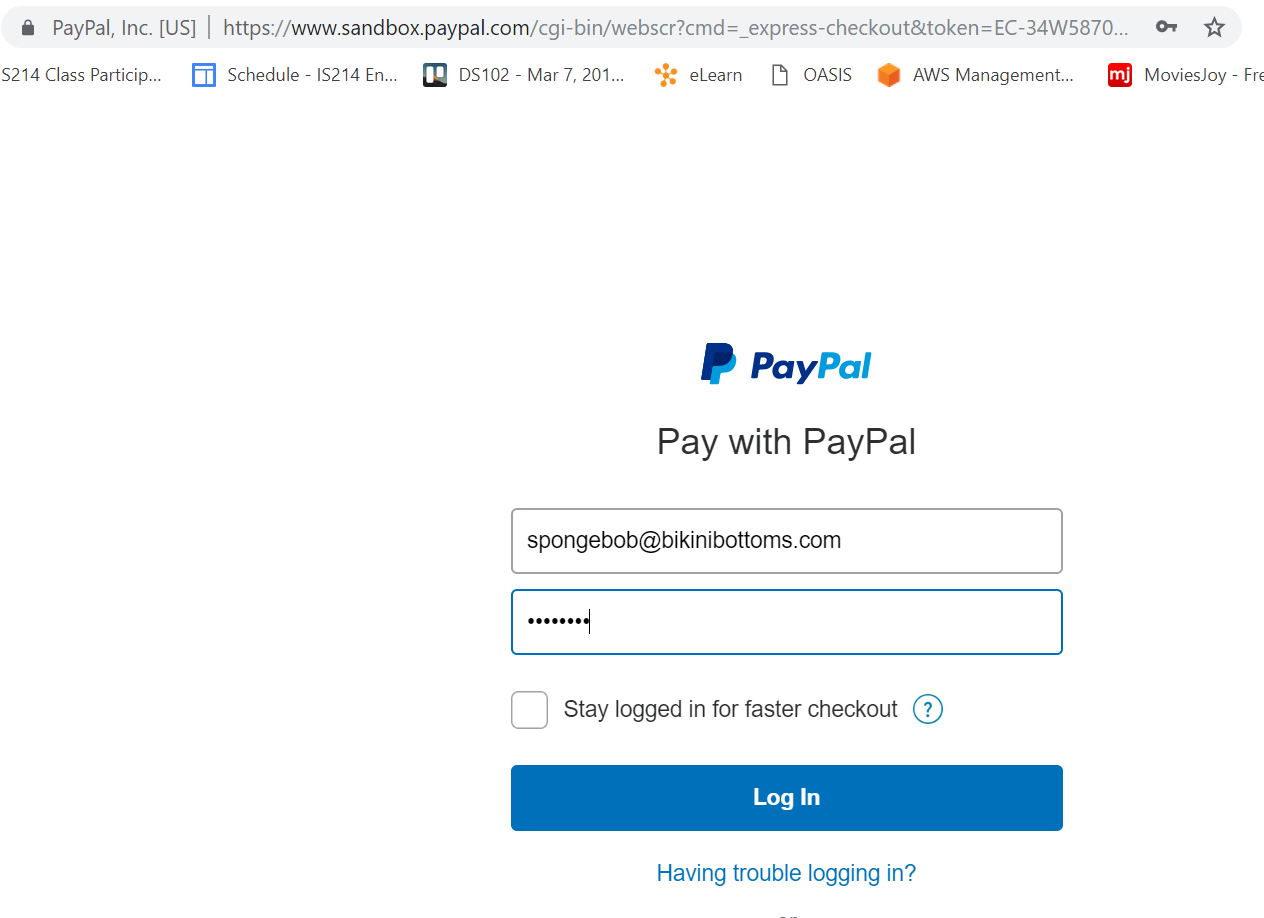


1. **The cart page retrieves the ‘Loyalty points’ for the customer and deducts it from the ‘Total Item price’ to display the ‘Total Payable Amount’.**

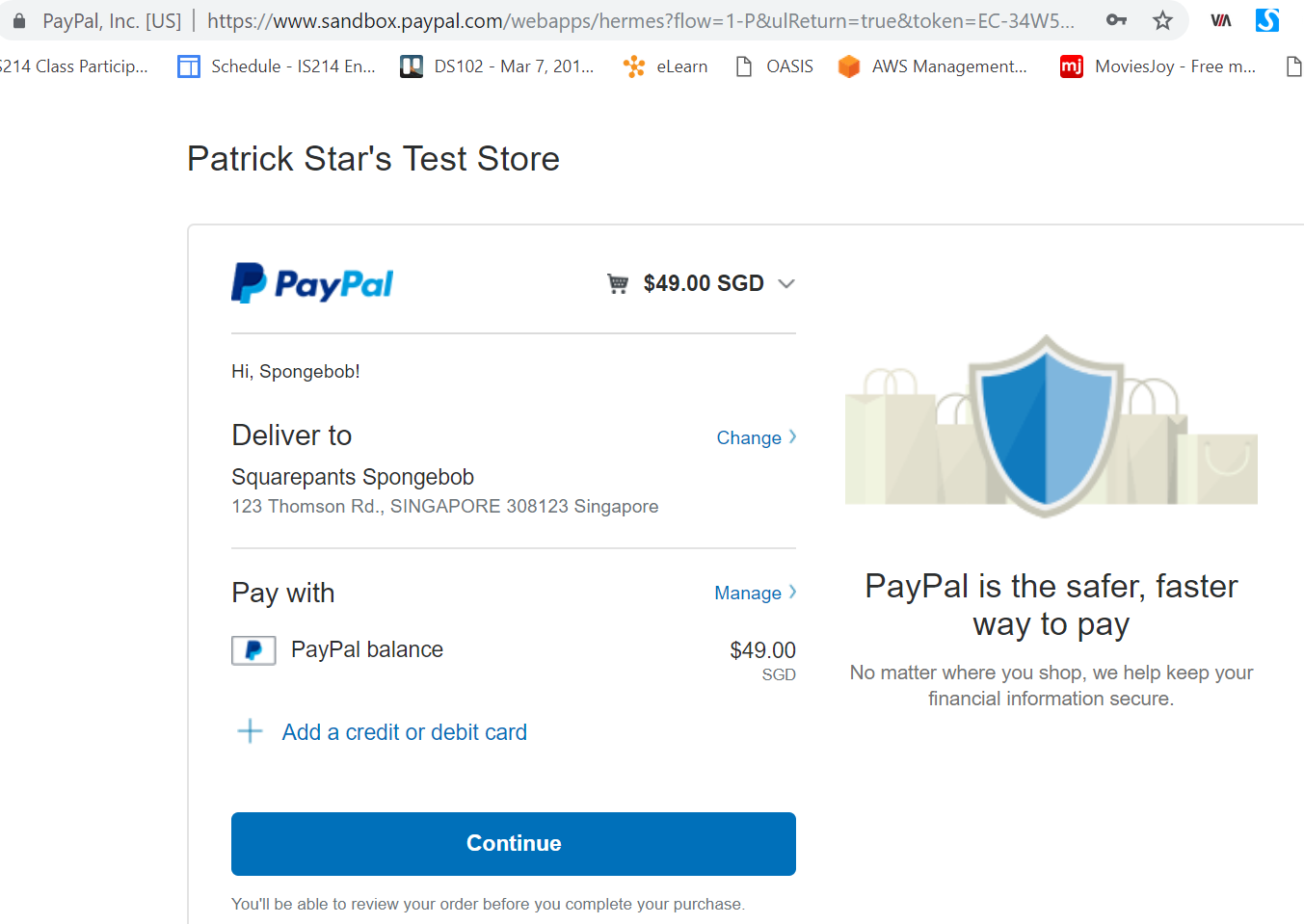
**[Scenario 2] To buy the items, the customer clicks on the ‘Proceed to Payment’ button.**



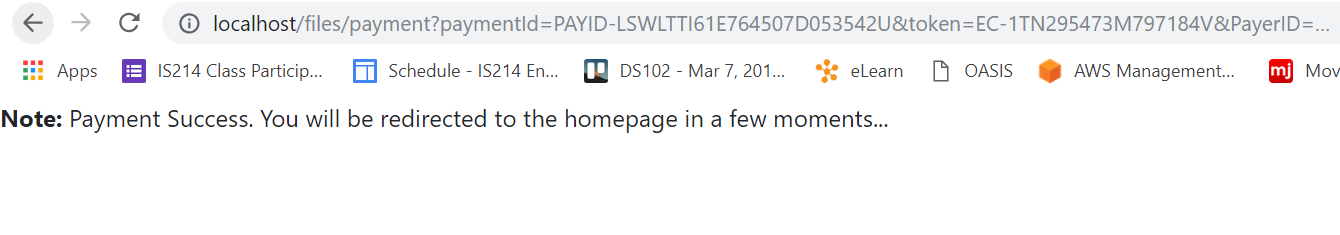
1. **Paypal log-in page is shown and customer logs in using the password “12345678”.**



1. **The Total payable amount is shown and the customer clicks on “continue” to complete the payment approval process.**



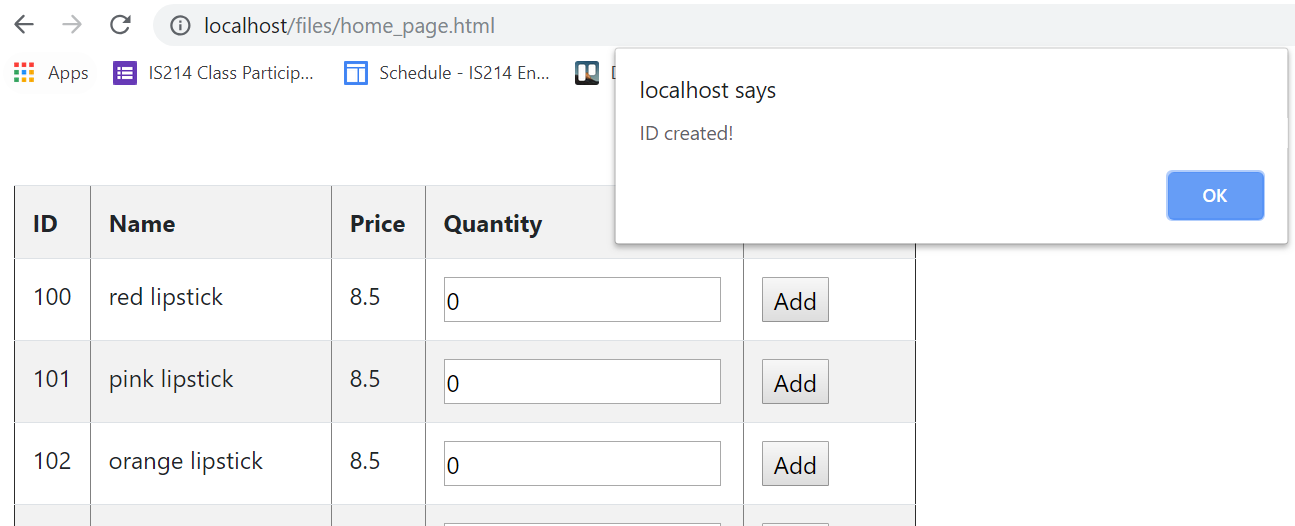
1. **The payment has been executed and the customer will be redirected back to the homepage shortly.**



*(To confirm that the payment has indeed been executed we can look at the console logs)*

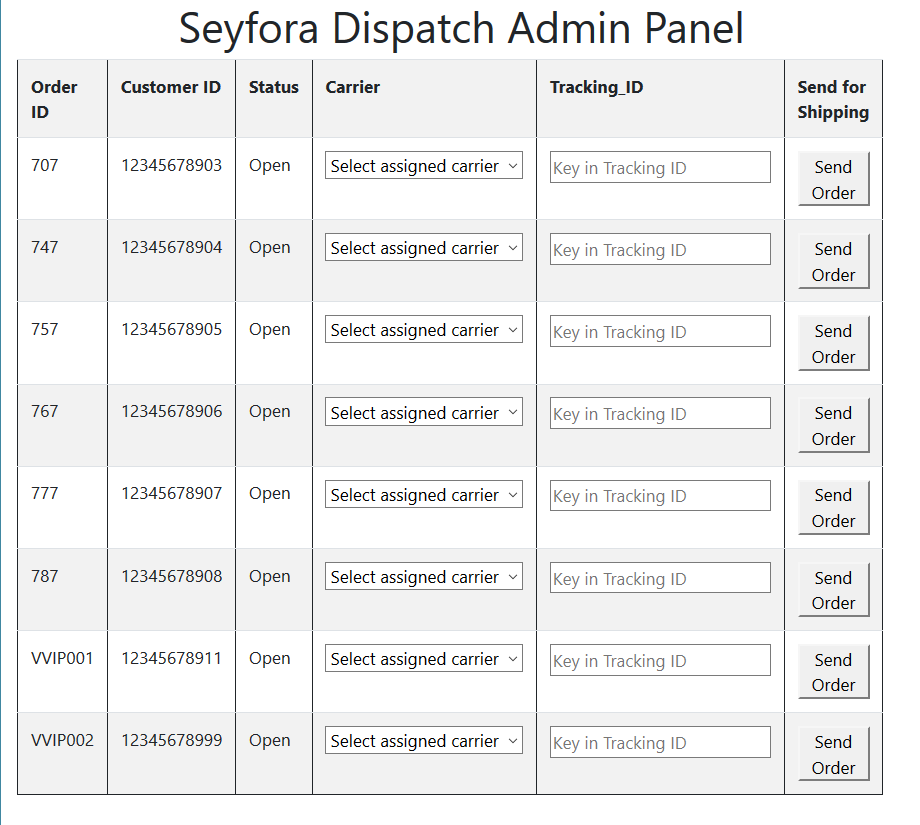


1. **Customer is now back at the homepage.**

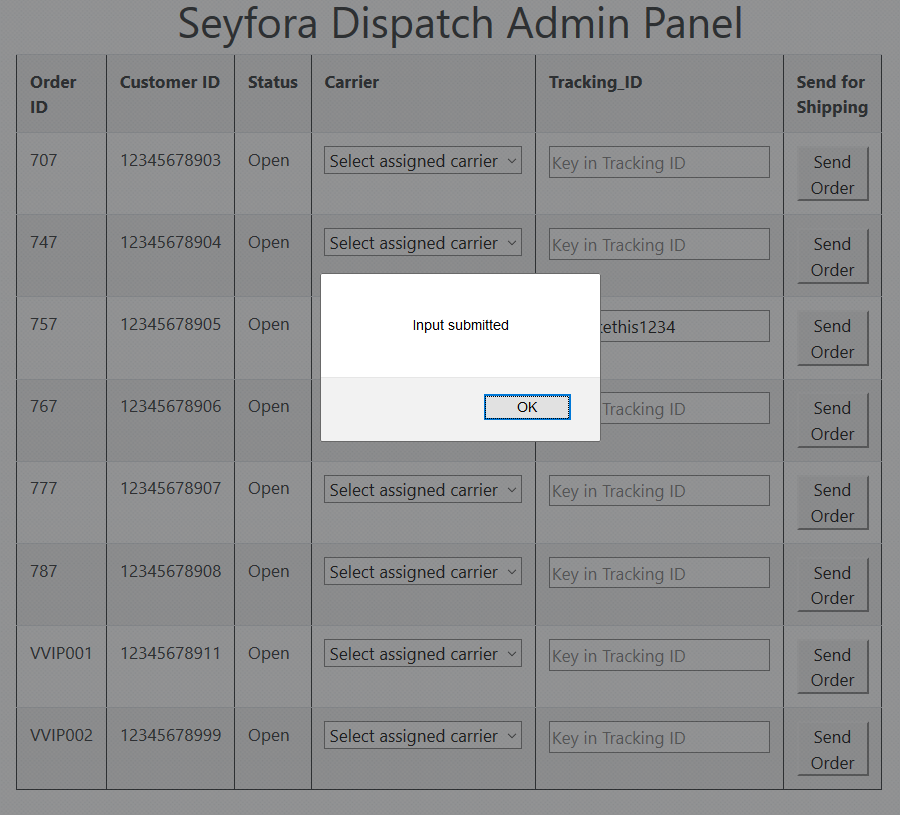


1. **[Scenario 3] Navigate to *http://<system name>/<designated webroot folder>/ship-admin.html***

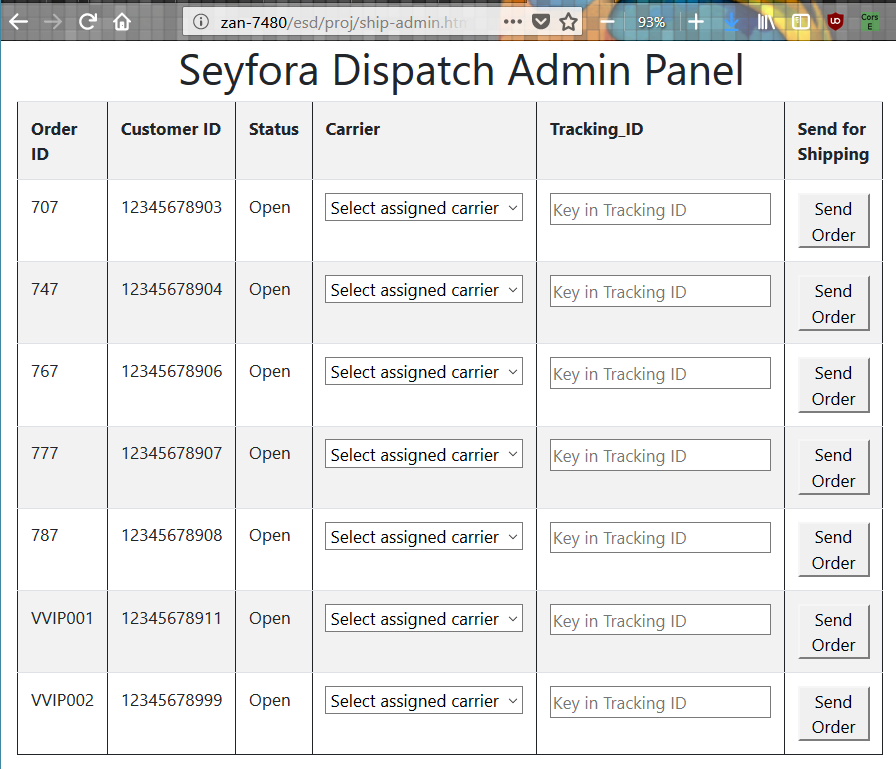
Here, all existing Orders with the “Open” status are listed. The user (staff) simply has to select a Carrier from the dropdown list and populate the Tracking ID field and click on the Send Order button.

**

1. **Submit Order for Shipment. Upon successful connection to backend services, a pop-up window will appear informing staff of successful input** *(here we Send Order 757 with Tracking\_ID: Whylikethis1234)*



1. **After acknowledging the pop-up notification, the page auto reloads and as the Order Status has been changed by the backend service from “Open” to “Shipped”, Order 757 is no longer listed.**



1. **Email address belonging to Customer of order #757 will receive a notification email as per existing details and information input by the staff in Step 10.**



**Note:** *VVIP001 recipient email is alanmegargel@smu.edu.sg and VVIP002 recipient email iseklum@smu.edu.sg*