Openpay Integration

Version 1.0.0

SDK Integration with Shop System

Table of Contents

1. OVERVIEW 3

1.1 COMPATIBILITY 4
1.2 LIMITATIONS, CONSTRAINTS 4
1.3 PRIVACY, PAYMENT 4

2. INTEGRATION OF SDK IN SHOP SYSTEM 5

2.1 INSTALLATION 5
2.2 INTRODUCTION TO LAYER APPROACH 5
2.2.1 Shop System 6
2.2.2 Business Layer 11
2.2.3 Api Layer 12

3. INTEGRATION OF SDK WITH SAP HYBRIS 13

3.1 FUNCTIONAL OVERVIEW 13
3.2 USE CASES 13
3.3 LIMITATIONS 13
3.4 COMPATIBILITY 13
3.5 IMPLEMENTATION GUIDE 14
3.5.1 Installation 14
3.5.2 Configuration of cron jobs 20
3.5.3 Testing of Plugin 22
3.5.4 Configuration of Openpay Icon and text for storefront using cmscockpit 34
3.5.5 Configuration of Openpay Icon and text for storefront using backoffice 37

4. KNOWN ISSUES 40

5. VERSIONS 40

1. Overview

This document describes the technical integration of Openpay link cartridge.

Openpay is a next generation payment solution that allows customers to buy now and pay over a flexible timeframe without interest, giving them added cash flow confidence. Creating a seamless omni channel experience (in-store, online and through our app), we service several brands across multiple industries including retail, healthcare, automotive & home improvement.

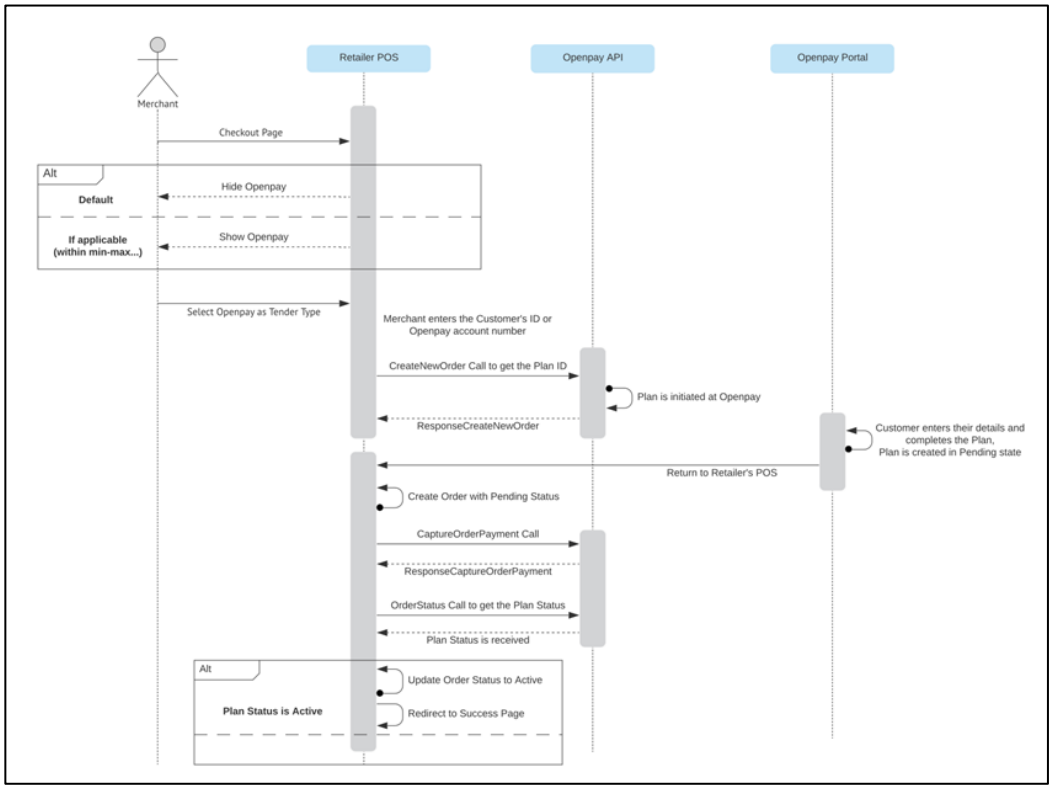
Our customers are empowered to live their best life; with more choice, time and flexibility.

The Openpay SDK is created with multiple layer approach which is further used to integrate with various Java shop systems.

1. Payment SDK includes multiple layers business layer, configuration layer, API layer as described below: -
    a. **Shop System: -** Java based shop system layer interacts with business layer of SDK, by sending request and do further implementations on getting the response back. For more detailed information how to integrate SDK with shop system refer to section 2.2.1
    b. **Business Layer: -** Business layer of SDK includes logic to create request data for each API method and make a call to API, send request and get the response back. This layer is created in generic way to be consumed by any Java based Shop System.
    c. **Configuration Layer: -** This is part of business layer of SDK includes three configuration files named as merchantMapping.ini, mappingShopConfig.ini and mappingApiConfig.ini under resources folder. But as we use SDK in form of jar these files are present at root level of jar.This layer basically handles all configurations like merchantConfig handles authentication related properties. Mapping shop and API config files handle attribute names , method names etc.
    d. **API Layer: -** This layer consists of making call to third party System (Openpay Payments) by sending required headers and body in json format and get the response back. Conversion of request to json, handling of authentication, connection creation all is handled in this layer.

2. Openpay Payment SDK is used in form of JAR file in respective shop system. This JAR file is provided to you by merchant.
3. SDK integration is done for SAP Hybris shop system and all related information is discussed in below chapter.

4. The below diagram represents flow between merchant shop and payment API. This is create order flow diagram which states how order data is sent from shop and using Openpay as checkout payment option order is successfully places in Openpay.



1.1 Compatibility

SDK is compatible with java version java 8 to java 11

1.2 Limitations, Constraints

None

1.3 Privacy, Payment

None

2. Integration of SDK in Shop System

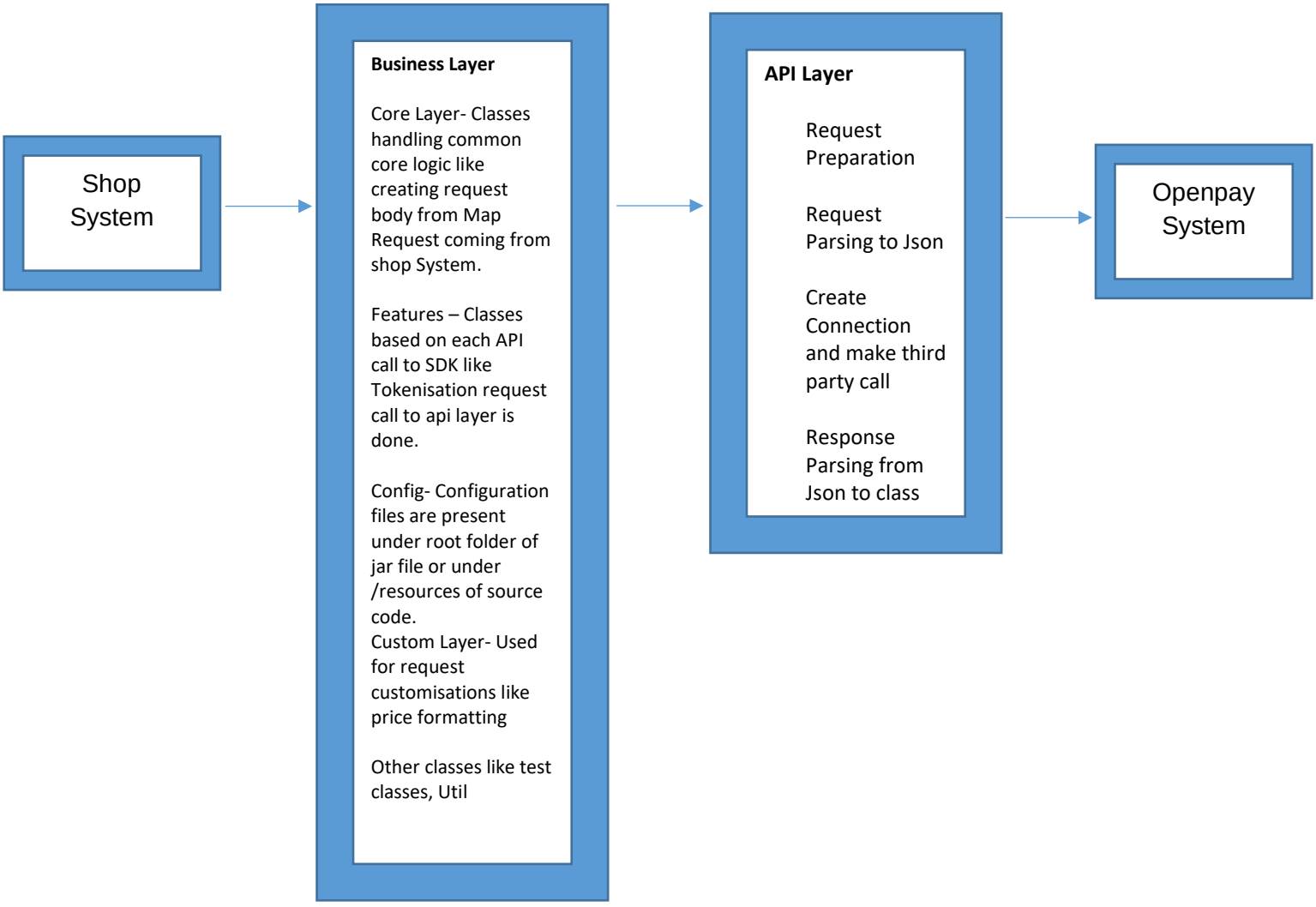
2.1 Installation

SDK is maven-based application created to consume all payment methods of Openpay. SDK is delivered in JAR format. That jar file of SDK will include all external libraries along with the layers we developed to consume the services and configuration files.

Include jar file to specific location of your shop system with other external dependencies to consume the jar methods.

2.2 Introduction to Layer approach

As already provided glimpse of SDK layer approach in overview section, here we will discuss about each layer in detailed way



2.2.1 Shop System

This layer refers to the specific shop system which we have to integrate with Openpay payment system. SDK will be provided in form of JAR.
Import JAR file in shop system where other external dependencies are located. All external libraries used in SDK is also included in JAR.

Integration of SDK with shop system

1. To consume all API's from SDK (refer Openpay documentation) to understand flow of various api's. Will be explaining how integration to be done with sdk using **Refund and Create New Order** API example.
2. To make refund from shop system will be sending refund related data to SDK system. As we can see in above flow diagram shop system will call business layer of SDK which is entry point for doing integration.
3. PlugintSDK.java is entry class for business layer and desired function call is made from shop system with required attributes to be sent.
4. For refund request below is function defination from PlugintSDK.java inside jar file

```
/**
 * Refund method creates request from @param refundData @param orderId and make
 * a call to api method in SDK layer. As this is a Generic method created to be
 * used on multiple platforms, Api method ([Method] section) and
 * class([ApiClass] section are loaded from mappingApiconfig.ini using
 * reflections
 *
 * Refund method basically used to do refunds
 *
 * @param refundData    It is a Map object coming from calling function which
 *                  contains relevant data required to make refund call like
 *                  refundData with key value pair map where keys can be
 *                  relevant to the attributes.Please refer API
 *                  documentation for fields related information {key,
 *                  value} : {String, Object}
 *
 * @param orderId       Object contains orderId/id's to get refund for
 *                  particular order. Refer API document to check data types
 *
 * @param attemptNumber Integer to be sent from calling function (of shop
 *                  system) with constant value as 1.This attribute is used
 *                  as counter to check number of attempts made for
 *                  retrying.
 *
```

Call the above function with required parameters, please refer to comments and Java Docs for the detailed explanation of function definition. Use the response for further implementations in shop system.

5. **As SDK is based on generic approach i.e. compatible with any Java based shop system, business layer plays around the logics to create request for each API call with data being sent from shop system.**

6. **Keys being used in parameter Map refundData has to be set as values in mappingShopConfig.ini located on root of Jar file under [Refund] section.**

For example:
refundData Map created is as follows
refundData = ["newPurchasePrice": $200.10
"reducePriceBy": 0
]

These keys set in refundData has to be set in Refund section of mappingShopConfig.ini as follows:

[Refund]
newPurchasePrice = newPurchasePrice
reducePriceBy = reducePriceBy

7. For create new order request below is function definition from PlugintSDK.java inside jar file

```
/**
        * GetToken method creates request from @param cartData and make a call to api
        * method in SDK layer. As this is a Generic method created to be used on
        * multiple platforms, Api method ([Method] section) and class([ApiClass]
        * section are loaded from mappingApiconfig.ini using reflections
        *
        * Tokenisation method basically creates a new order request
        *
        * @param cartData      It is a Map object coming from calling function which
        *                      contains relevant data required to make create new order
        *                      call with key value pair map where keys can be relevant
        *                      to the attributes.For creating new order you can send
        *                      cart data object(if it contains all relevant information
        *                      else send data based on shop system) in map along with
        *                      other fields like callbackURL,failURL etc Please refer
        *                      API document to check attributes related information
        *                      Please refer API documentation for fields related
        *                      information {key, value} : {String, Object}
        *
        * @param attemptNumber Integer to be sent from calling function (of shop
```

Call the above function with required parameters, please refer to comments and Java Docs for the detailed explanation of function definition. Send all fields required to make this API call in cartData Map from shop system. Use the response for further implementations in shop system example fetch transaction token from response map and redirect to Openpay using redirectURL which can be stored in property file of shop system.

8. Keys being used in parameter Map cartData has to be set as values in mappingShopConfig.ini located on root of Jar file under [createOrder] section.

For example cartData Map sent from shop system created is as follows

{totalItems=2, redirectUrl=https://uat-mrt.magentaretail.com.au/, code=00015001, totalPrice={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$2,522.87, value=500.98}, origin=ONLINE, failUrl=https://uat-mrt.magentaretail.com.au/, subTotal={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$2,513.88, value=500.88}, planCreationType=Pending, customerQuality=-1, emailAddress=ecommerce@xx.yy, entries=[{updateable=true, product={purchasable=true, code=1432722, name=Gigashot K80H, description=null, url=/Open-Catalogue/Cameras/Hand-held-Camcorders/Gigashot-K80H/p/1432722}, statusSummaryMap={}, quantity=1, totalPrice={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$396.88, value=396.88}, deliveryPointOfService=null, cancellableQty=0, cancelledItemsPrice=null, entries=null, deliveryMode=null, entryNumber=0, configurationInfos=[], basePrice={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$396.88, value=396.88}}, {updateable=true, product={purchasable=true, code=898503, name=1V, description=null, url=/Open-Catalogue/Cameras/Film-cameras/1V/p/898503}, statusSummaryMap={}, quantity=1, totalPrice={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$2,117.00, value=2117.11}, deliveryPointOfService=null, cancellableQty=0, cancelledItemsPrice=null, entries=null, deliveryMode=null, entryNumber=1, configurationInfos=[], basePrice={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$2,117.00, value=2117.11}}], deliveryMode={code=standard-gross, name=Standard Delivery, description=3-5 business days, deliveryCost={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$8.99, value=8.99}}, deliveryAddress={lastName=Gupta, country={isocode=BY, name=Belarus}, town=Bangalore, companyName=null, postalCode=560048, title=Dr., titlefirstNameCode=dr, firstName=Nupur, formattedAddress=14171 prestige shantiniketan, Whitefield, Bangalore, 560048, phone=09740755388, visibleInAddressBook=true, shippingAddress=true, id=8796650110999, billingAddress=true, region=null, line2=Whitefield, line1=14171 prestige shantiniketan, email=ecommerce@xx.yy, defaultAddress=false}, chargeBackCount=-1, totalPriceWithTax={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$2,522.87, value=500.98}, net=false}

CartData map sent from shop system, above example is based on Hybris shop system where as it is cartData object is set to cartData map along with other mandatory fields like origin,redirectUrl etc. Optional fields can be left as blank as you can see below like dob if not available. Also, above request map is nested map, below config values are set in (.) pattern below like for resLine1 will set value as deliveryAddress.phone. If we have array in request map then for those entries use 'array' (prefix before field name) as text after main key, ex. for product data we have entries array so, values below will be set as itemName = entries.array.product.name

Also in mappingShopConfig.ini in section [PriceFormattingFields] and key priceFormattingFields add all attribute names which handle price value like purchasePrice in shop is coming as pu**rchasePrice = totalPrice.value.** So set totalPrice.value in the priceFormattingFields key. Other values are also added separated by (,)

```
[createOrder]
origin = origin
redirectUrl = redirectUrl
cancelUrl = failUrl
failUrl = failUrl
planCreationType = planCreationType
chargeBackCount = chargeBackCount
customerQuality = customerQuality
firstName = deliveryAddress.firstName
middleName =
surName = deliveryAddress.lastName
email = emailAddress
dob =
gender =
phoneNumber = deliveryAddress.phone
resline1 = deliveryAddress.line1
resline2 = deliveryAddress.line2
rescity = deliveryAddress.town
resstate = deliveryAddress.town
respincode = deliveryAddress.postalCode
delline1 = deliveryAddress.line1
delline2 = deliveryAddress.line2
delcity = deliveryAddress.town
delstate = deliveryAddress.town
delpincode = deliveryAddress.postalCode
deliveryDate =
description = deliveryMode.description
purchasePrice = totalPrice.value
retailerOrderNo = code
itemName = entries.array.product.name
itemGroup = entries.array.product.name
itemCode = entries.array.product.code
itemGroupCode = entries.array.product.name
itemRetailUnitPrice = entries.array.basePrice.value
itemQty = entries.array.quantity
itemRetailCharge = entries.array.totalPrice.value
```

NOTE : Don't change keys or section names of mappingShopConfig.ini while doing changes as these keys are mapped to keys of mappingApiConfig.ini

9. **The above steps are main steps used to integrate SDK JAR with shop system.**

10. **Add properties like Openpay redirect URL (provided by Openpay) , callbackURL, failURL (i.e. data needs to be handled from shop system) in shop system property file.**

11. **Price data for various API has to be sent with some denomination value (description provided in API doc), this price formatting is already handled by SDK for request related data.**

12. **For response related data please handle price formatting in shop system i.e. for API call { {/orders/limits}**

13. **For various API calls, which functions to call from PlugintSDK.java in businessLayer of JAR is as follows in below table. Please follow java docs for reference to function definition.**

API calls from Openpay	Function Definition to be referred from SDK
GET{/orders/{orderId}	public Map<String, Object> updateShopOrder(final Object orderId, int attemptNumber) throws Exception();
POST {/orders}	public Map<String, Object> getToken(final Map<String, Object> cartData, int attemptNumber) throws Exception();
POST {/orders/{orderId}/capture}	public Map<String, Object> capturePayment(final Object orderId, int attemptNumber) throws Exception();
POST {/orders/{orderId}/refund}	public Map<String, Object> refund (final Map<String, Object> refundData, final Object orderId, int attemptNumber) throws Exception ();
GET {/orders/limits}	public Map<String, Object> getPSPConfig(int attemptNumber) throws Exception();

14. **Initiating Token call from shop and redirecting to Openpay. Below example is just algorithm, please use logic based on your shop system like getting cart data.**

```
public void sendReuestToCreateNewOrder() {

//get cartData from shop

final CartData cartData = getCheckoutFacade().getCheckoutCart();

Map cartDataMap = omapper.convertValue(cartData, Map.class);

//put the other required attributes in property file and read the values from property file

cartDataMap.put("origin", getConfigurationService().getConfiguration().getProperty("origin"));

cartDataMap.put("redirectUrl", getConfigurationService().getConfiguration().getProperty("redirectUrl"));

cartDataMap.put("failUrl", getConfigurationService().getConfiguration().getProperty("failUrl"));

cartDataMap.put("planCreationType", getConfigurationService().getConfiguration().getProperty("planCreationType"));

cartDataMap.put("chargeBackCount",
Integer.parseInt(getConfigurationService().getConfiguration().getProperty(OpenpaypaymentaddonConstants.

CHARGE_BACK_COUNT).toString()));

cartDataMap.put("customerQuality", Integer.parseInt(getConfigurationService().getConfiguration()
        .getProperty(OpenpaypaymentaddonConstants.CUSTOMER_QUALITY).toString()));

final Map<String, Object> createTokenResponse = getPlugintSdk().getToken(cartDataMap,1);

//getting transaction token

//create a function to get transaction token from response by iterating the response

final String transactionToken = getTransactionToken(createTokenResponse,
getConfigurationService().getConfiguration().getProperty(OpenpaypaymentaddonConstants.REDIRECT_TOKEN_FIELD_NAME).toString())
;

//fetch redirect url provided by Openpay from property file

final String redirectUrlToOpenPay = getConfigurationService (). getConfiguration(). getProperty("OpenPayRedirectUrl").toString();

//redirect URL to browser using ajax call

return redirectUrlToOpenPay + transactionToken;

        }
```

2.2.2 Business Layer

This section will explain what all is availble in jar for this section. Except config files no file can be changed under JAR. As explained in diagram under section 2.2 business layer consist of multiple packages like

1. com.plugint.businessLayer.constant
2. com.plugint.businessLayer.core
3. com.plugint.businessLayer.features
4. com.plugint.businessLayer.util
5. com.plugint.customLayer
6. Config files which are present on /root of JAR and /resources of source code

For detailed understanding of all classes under above packages please refer to Java docs. Config file section will be explained in below section. PlugintSDK.java under com.plugint.businessLayer.core is the entry point for integration already explained in section 2.2.1

2.2.2.1 Configuration Section

This section is part of business layer and main logic resides here. There are 3 configuration files named as

1. **merchantConfig.ini** – This file is basically for merchant to add configuration properties like authentication details. Any changes related to authentication are done under [Authentication] section.
2. **mappingApiConfig.ini** – This file consist of all API related properties like host name. This file should not be changed until there is some changes in API document. As business layer uses Java Reflections method names, api class names , api request model names are present under this file.
3. **mappingShopConfig.ini** – This file consist of all shop related properties as discussed in point 6 of section 2.2.1. This file will be changed for every new Java based shop system. But only values need to be changed based on request sent from shop system, already explained above in point 6 of section 2.2.1.

If any new API method gets added then new section is introduced in both mappingShopConfig.ini and mappingApiConfig.ini with keys as common for both files.

2.2.2.2 Custom Layer

This layer is part of business layer created to customise attributes for request. Taking an example, need to send prices to Openpay in certain format,i.e. multiply each price value by 100 and send it. These kind of customisations are handled in this layer.

The changes need to be made is in mappingShopConfig.ini in section [PriceFormattingFields] and key priceFormattingFields. For this particular field add all attribute names which handle price value like purchasePrice in shop is coming as pu**rchasePrice = totalPrice.value.** So set totalPrice.value in the priceFormattingFields key. Other values are also added separated by (,).

2.2.3 Api Layer

This section is interacted by Business Layer. It basically contains all the logic related to client side interatcion like parsing json request, handling headers, creating connection and parsing response back to java objects. This layer basically contains multiple packages listed below

1. com.plugint.client
2. com.plugint .client.api
3. com.plugint.client.auth
4. com.plugint.client.model

For detailed understanding of all java classes present under above packages please refer java docs. Java documentation is present before each function inside class and separate folder is created under source code /target/apidocs/com/plugint.

OrderApi.java is main API class being called from business layer. As business layer uses Java Reflections method names, api class names everything is present under mappingApiConfig.xml

Below table represent the method call done to API layer (OrdersApi.java) from business layer(PlugintSDK.java).

Function Definition to be referred from SDK	Function of API Layer
public Map<String, Object> updateShopOrder(final Object orderId, int attemptNumber) throws Exception();	**public IntegrationApiModelsOrder ordersOrderIdCancelPost(String orderId) throws ApiException();**
public Map<String, Object> getToken(final Map<String, Object> cartData, int attemptNumber) throws Exception();	**public IntegrationApiModelsOrder ordersPost(IntegrationApiModelsCommandsCreateOrder body) throws ApiException();**
public Map<String, Object> capturePayment(final Object orderId, int attemptNumber) throws Exception();	**public IntegrationApiModelsOrder ordersOrderIdCapturePost(String orderId) throws ApiException();**
public Map<String, Object> refund (final Map<String, Object> refundData, final Object orderId, int attemptNumber) throws Exception ();	**public IntegrationApiModelsOrder ordersOrderIdRefundPost(String orderId, IntegrationApiModelsCommandsRefund body) throws ApiException();**
public Map<String, Object> getPSPConfig(int attemptNumber) throws Exception();	**public IntegrationApiModelsLimits ordersLimitsGet() throws ApiException ();**

Above API functions are handled in business layer using Java Reflections. If there are any changes to API documentation like change in method name or change in API models, please refer to mappingAPIconfig.xml. This file contains separate section for each type like for API models → [ApiModels] section has to be changed

3. Integration of SDK with SAP Hybris

The Openpay – SAP Hybris Commerce Cartridge allows seamless integration between SAP Hybris Commerce and Openpay payment gateway. New option is added in Hybris checkout flow to pay with Openpay under certain conditions.

3.1 Functional Overview

This cartridge includes new payment method as Openpay in checkout flow under payment section. This section is only enabled if cart lies under specific price range, which is configured in Hybris Configuration section. Successful purchase flow can be done through Openpay payment method, cancel flow and refunds can also be handled.

3.2 Use Cases

1. Order can be placed successfully using Openpay payment method configured under payments section of Checkout flow
2. If there is some problem occurred during purchase flow, cancel option is also available which will take you back to cart page and no order is created in Hybris
3. Once order is successfully purchased from Openpay, payment is captured in Openpay then order status is updated in Hybris Order Model.
4. If by mistake order is not captured in Openpay payment gateway, then automatic cron job configured in Hybris backoffice will run after every 30 minutes to check order status.
5. Openpay price limits are configured as well using cron job. This job will run every day at 12:00 PM to check the min max price limits and update in configuration section of HAC and write the values in local.properties as well
6. Refund functionality is also available for orders paid with Openpay in backoffice. Under Order type new refund tab is created which is only visible for orders paid with Openpay.

3.3 Limitations

For newer Hybris versions Hybris CMSCockpit might not be available for components configured like Openpay payment logo, error message. This can be configured from backoffice as well

3.4 Compatibility

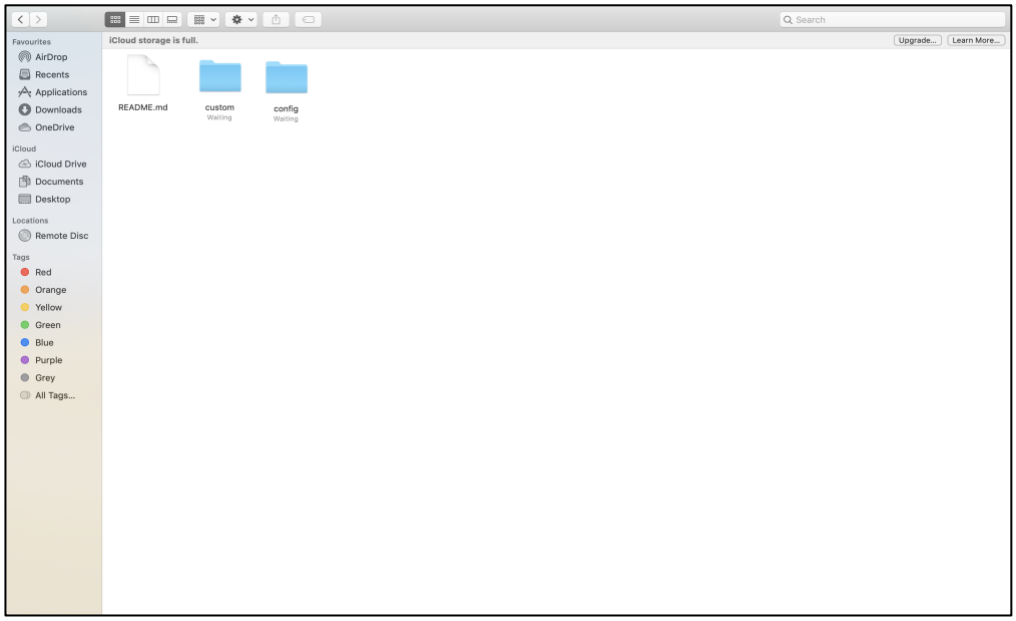
Openpay payment cartridge is built in Hybris 6.2.0. Migration is required to use the cartridge in upper versions. Not compatible with lower versions. This cartridge is built for demo stores i.e. **apparel-uk, electronics and apparel-de** of **B2C** accelerator. All UI components are used OOTB.

3.5 Implementation Guide

3.5.1 Installation

Installation of Openpay payment cartridge in SAP Hybris platform is as follows-

- Already assumed that Hybris is installed on machine, if not please install Hybris suite with B2C accelerator recipes
- Fetch the plugin required extensions (either a zip file or take a pull from GIT repo if available)
- The code structure will look like



- If working on new Hybris suit then copy entire custom folder under {Hybris_Dir}/hybris/bin, else if its old suite and custom folder is already available just copy the extensions to the already created custom folder
- You can create symbolic link for custom folder as well from repository to Hybris/bin so that updated code can be reflected automatically.
- Copy to content of {gitRepo}/config/customize to {HYBRIS_DIR}/hybris/config/customize.
- Copy content of files from config folder and copy them under {Hybris_Dir}/hybris/config.

LocalExtensions.xml

```
<!-- Open Pay Integration-->

<extension name='openpaypaymentaddon'
/> <extension name='openpayintegration' />
<extension name='openpaybackofice' />
```

Local.properties : Copy these content to local.properties file of you Hybris suite. Please change the domain name for redirectUrl and failUrl according to your domain if any.

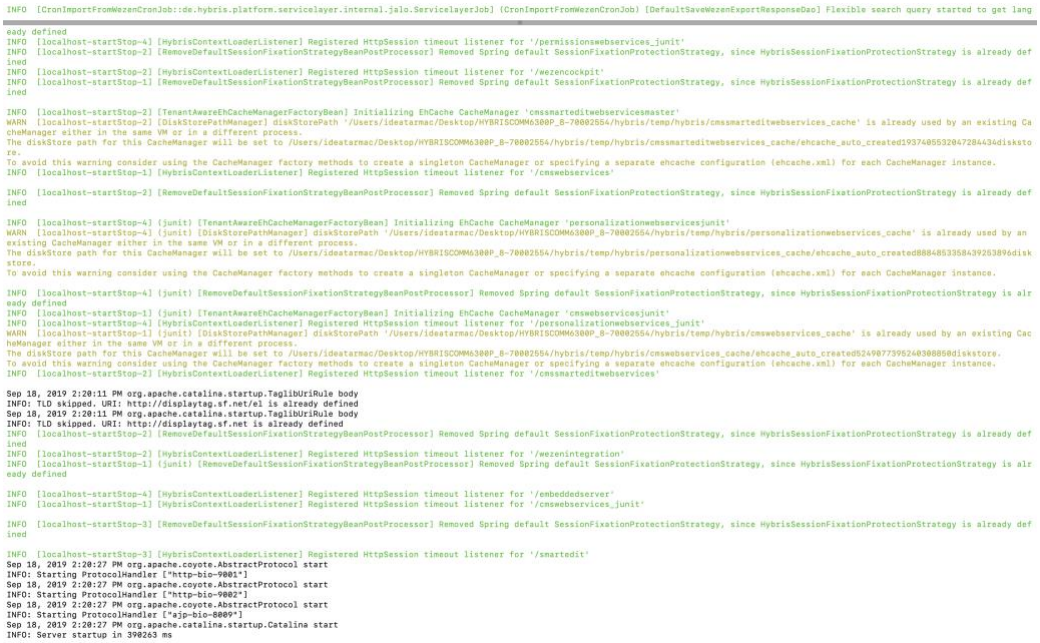
```
redirectTokenFieldName=TransactionToken

planCreationType=Pending

redirectUrl=https\://localhost\:9002/yacceleratorstorefront/checkout/multi/payment-method/placeOrderWithOpenPay

failUrl=https\://localhost\:9002/yacceleratorstorefront/cart

customerQuality=-1

chargeBackCount=-1

OpenPayRedirectUrl=https\://retailer.myopenpay.com.au/websalestraining/?TransactionToken\=

origin=ONLINE

jarResourceCmsValue=jar:com.openpay.setup.OpenpayPaymentAddonInitialDataSetup&/openpaypaymentaddon/import/cockpits/cmscockpit

maxRetry = 3
```

Now please update the domain name for failrUrl and redirectUrl, if using localhost then fine, if using something else, please update it and save the file.This can be changed from HAC as well but to persist it after server restart please update to this file as well.

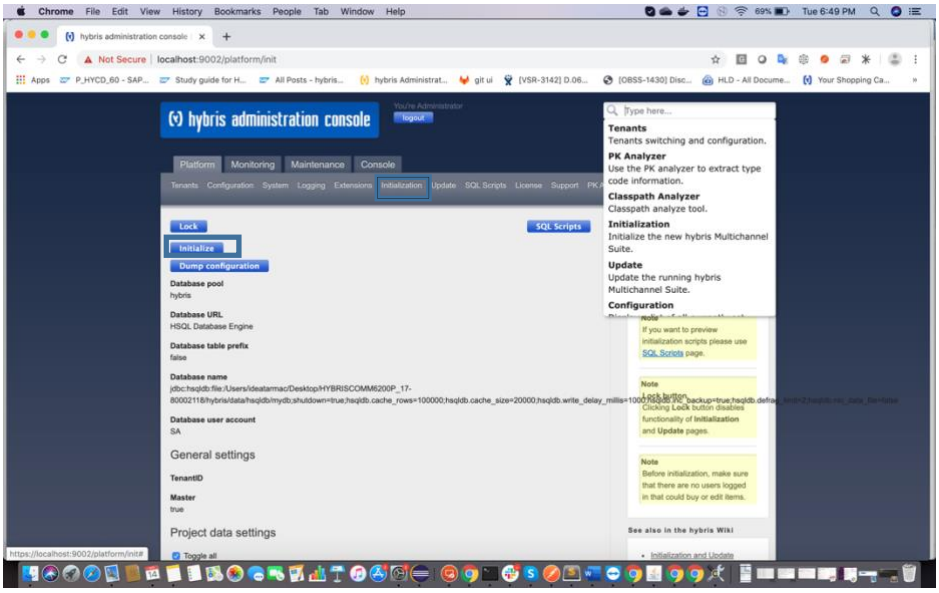
Windows	**MAC/Linux**
setantenv.bat	. ./setantenv.sh
ant customize	ant customize
ant addoninstall -Daddonnames=’openpaypaymentaddon’ -DaddonStorefront.yacceleratorstorefront="yacceleratorstorefront"	ant addoninstall -Daddonnames=’openpaypaymentaddon’ -DaddonStorefront.yacceleratorstorefront="yacceleratorstorefront"
ant clean all (after success)	ant clean all (after success)
hybrisserver.bat	./hybrisserver.sh

- Now go to / {Hybris path}/hybris/bin/platform open cmd and run following commands to build the plugin

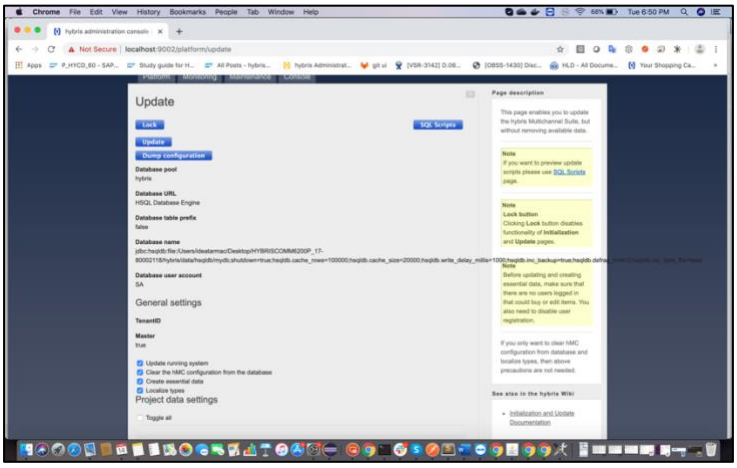
- Once ./hybrisserver.sh is completed with success it looks like –

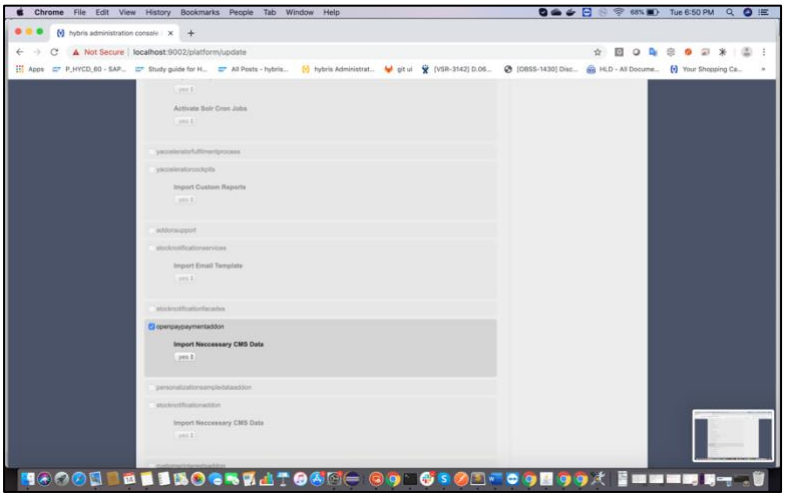


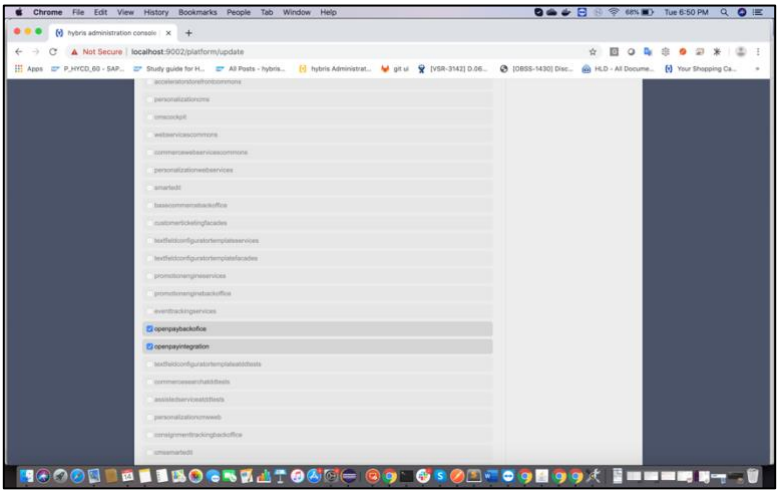
- If its new Installation(installing hybris for the first time) just go to Browser and type URL **{https://localhost:9002/}** login with credentials **admin/nimda**. Go under Platform tab --> Initialization and press button **Initialize** and then perform the next step as well**. If already initialized once jump to next step directly.**



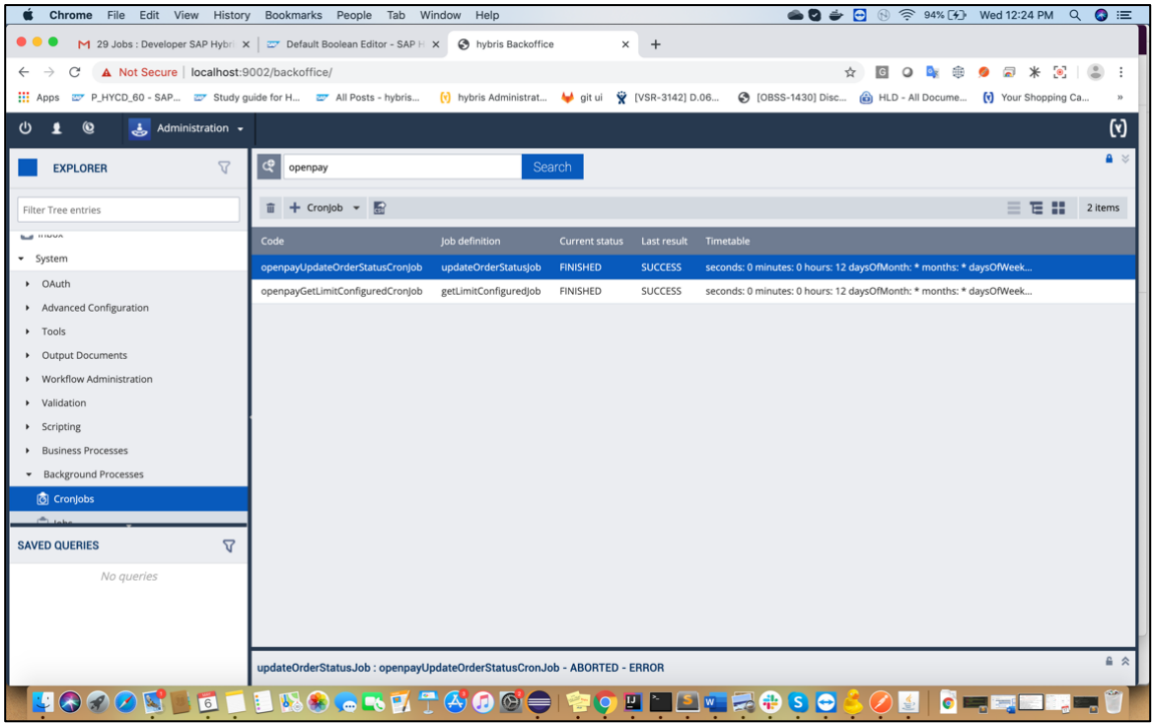
- Go to Browser and type URL {https://localhost:9002/} login with credentials admin/nimda. Go under Platform tab --> Update, select openpayintegration,openpaybackoffice amd openpaypaymentaddon and run update.





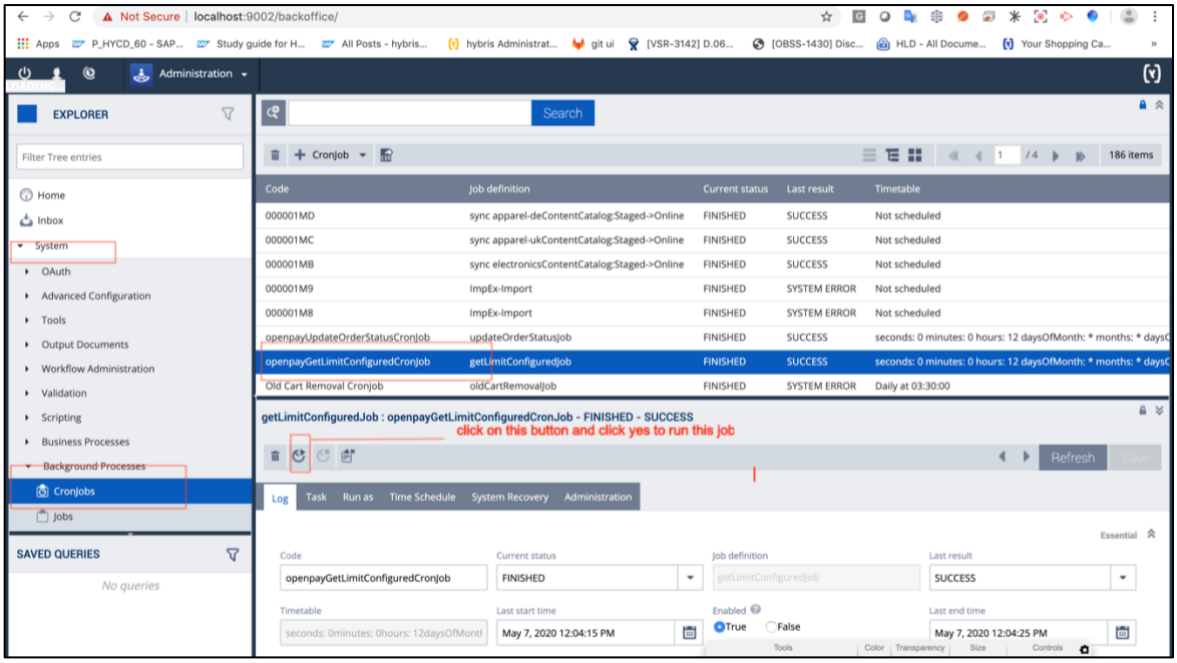


- **Once completed open backoffice by URL https://localhosthost:9002/backoffice. Check whether cron jobs are added or not by going to System->Background Processes -> Cronjobs Automatic cronjob for limits is configured which has to be run once manually during installation**

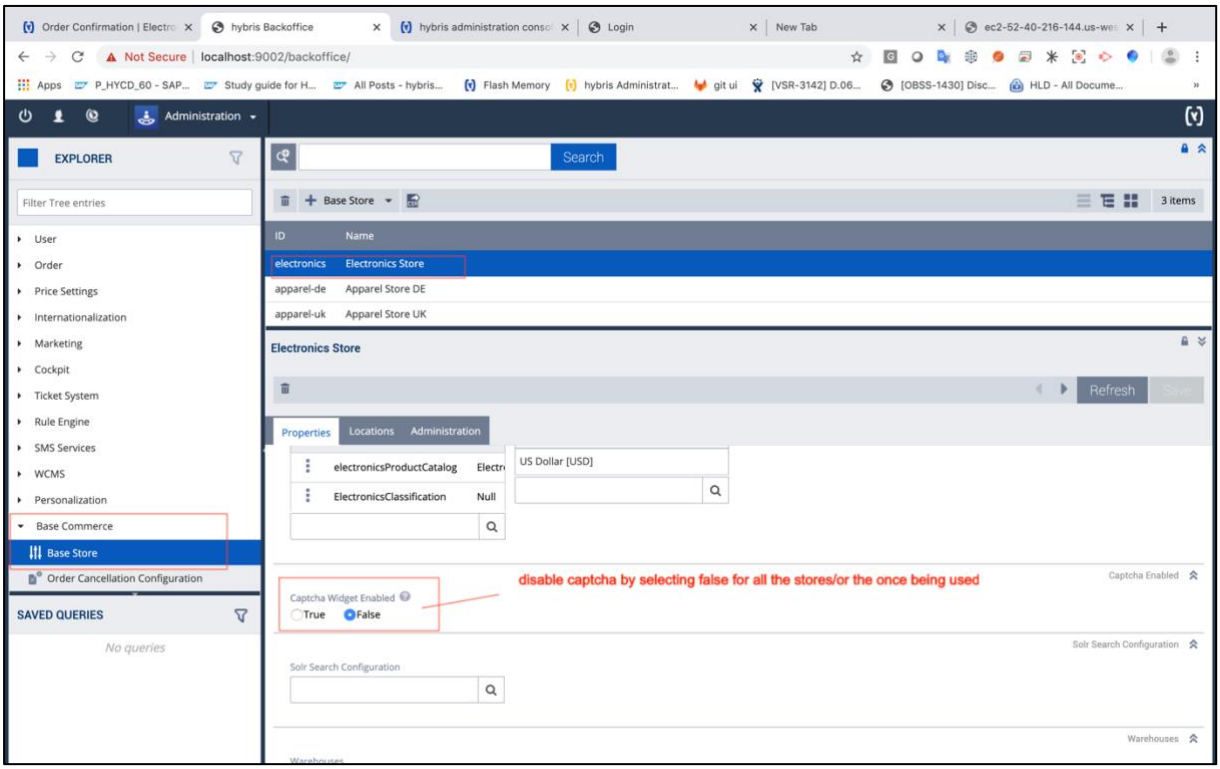


Now cartridge is installed successfully.

- **Select the openpayGetLimitConfiguredCronJob and run the cron job once as it if first time installation. Go to Backoffice ([https://localhost:9002/backoffice) -->](https://localhost:9002/backoffice)) Login with credentials (admin/nimda) → System tab → BackgroundProcess →CronJobs → Select cron job openpayGetLimitConfiguredCronJob.**



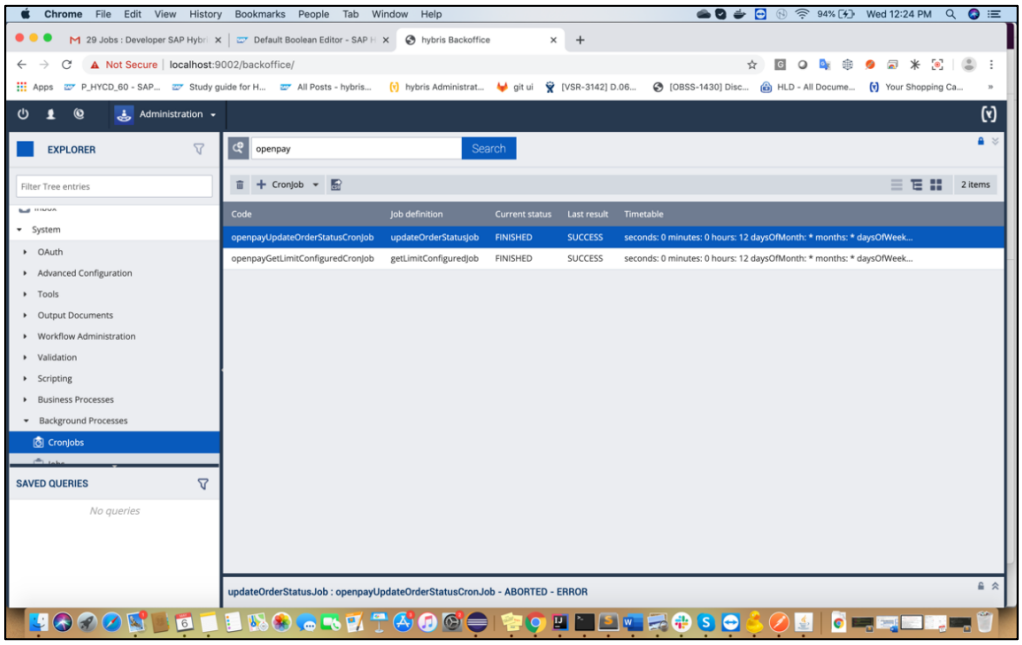
- **If its new installation for Hybris 6.2 register does not work OOTB due to some recaptcha private and public key, so please just disable the captcha from backoffice and then registration works. We can checkout as guest as well. This is just in case registration is required.**



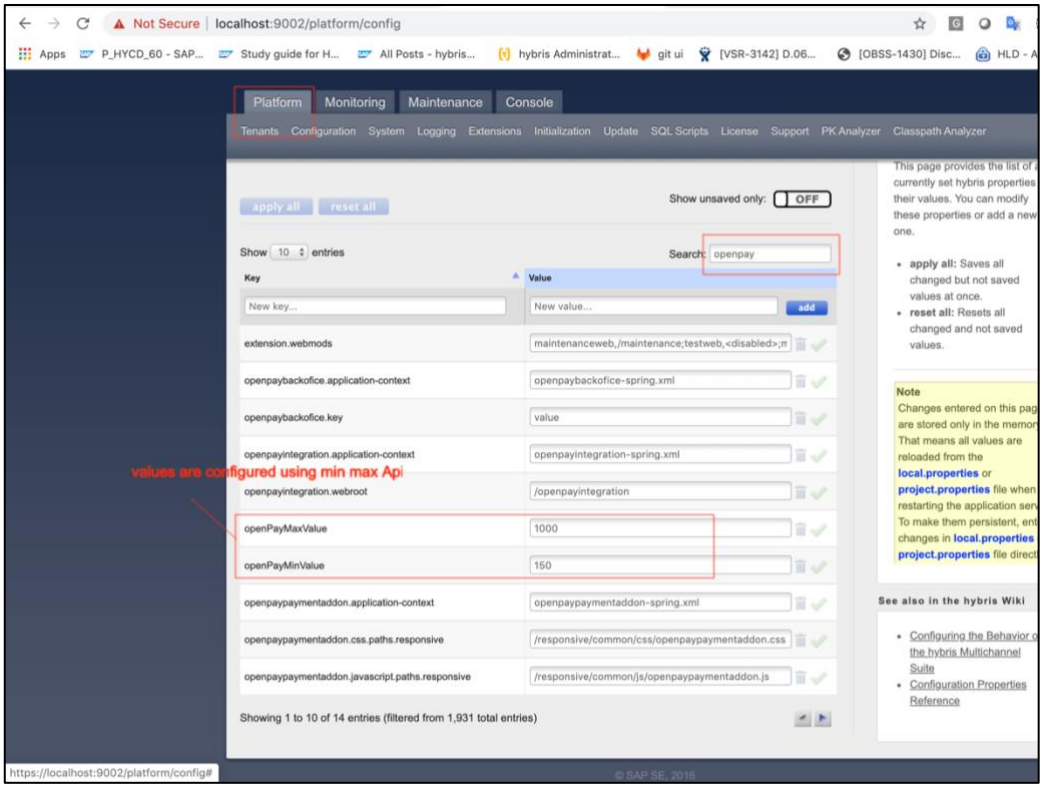
NOTE : If Hybris return functionality is used OOTB please replace {Hybris_Dir}/hybris/config/customize/ OOTB_REFUNDS/ CaptureRefundAction.java with OOTB refund action class in fulfilmentprocess or ordermanagement extension for using OOTB returns for Openpay as well. Second add dependency of openpayintegration to your fulfilmentprocess or ordermanagement extension.

3.5.2 Configuration of cron jobs

As discussed in last step two cron jobs are configured in backoffice after successful installation of payment plugin.



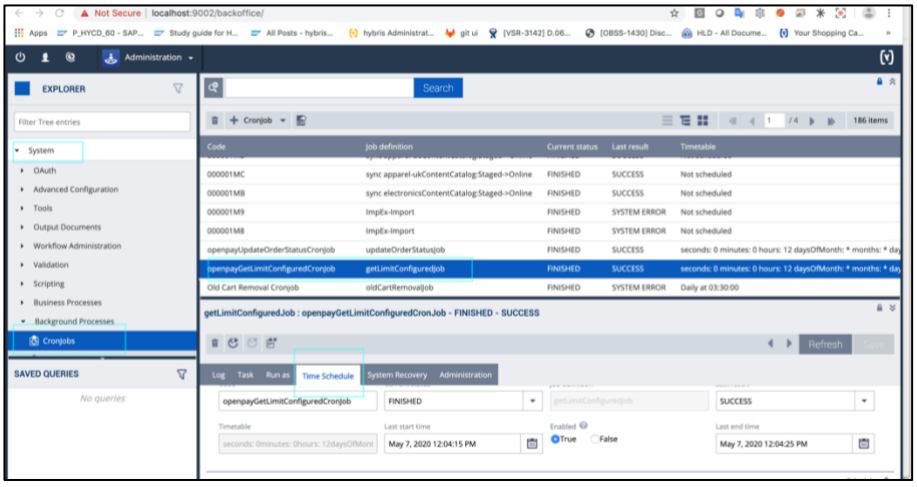
openpayGetLimitConfiguredCronJob – Cron job configured the minimum and maximum limit price to pay order using Openpay payment method. This cron jobs mainly makes a call to SDK layer and get the price range. For first time plugin installation this job has to be configured manually once. It's an automatic job which runs every day at 12:00 PM.
Once limits are configured they can be checked in Hybris Administration control. Go to https://localhost:9002 → login using password (nimda)→ Go to Platform tab →select configuration tab → search with keyword 'Openpay'



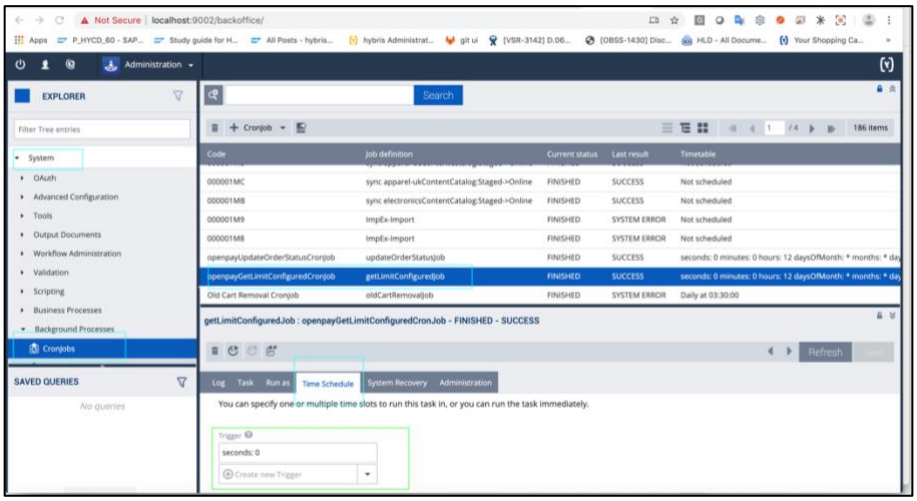
openpayUpdateOrderStatusCronJob - Cron job is configured to check the status of orders paid with Openpay but the payment status is in pending state. This cronjob is currently configured with trigger of running everyday at 12:00 PM.

Triggers can be changed from backoffice for cron jobs, please check the steps below

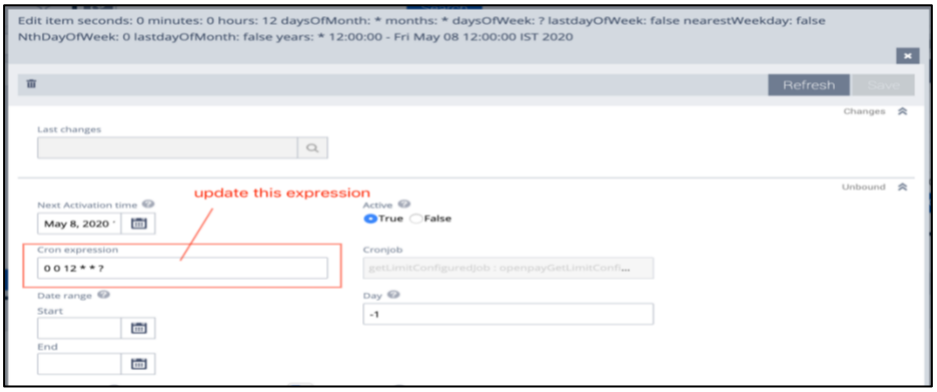
- **Open particular cronjob to update the scheduled trigger by Go to Backoffice (https://localhost:9002/backoffice) --> Login with credentials (admin/nimda) → System tab → BackgroundProcess →CronJobs → Select one cron job to be updated like openpayUpdateOrderStatusCronJob. Go to highlighted tab Time Schedule**



- **Go to trigger attribute in Time Schedule tab and double click the already added trigger.**

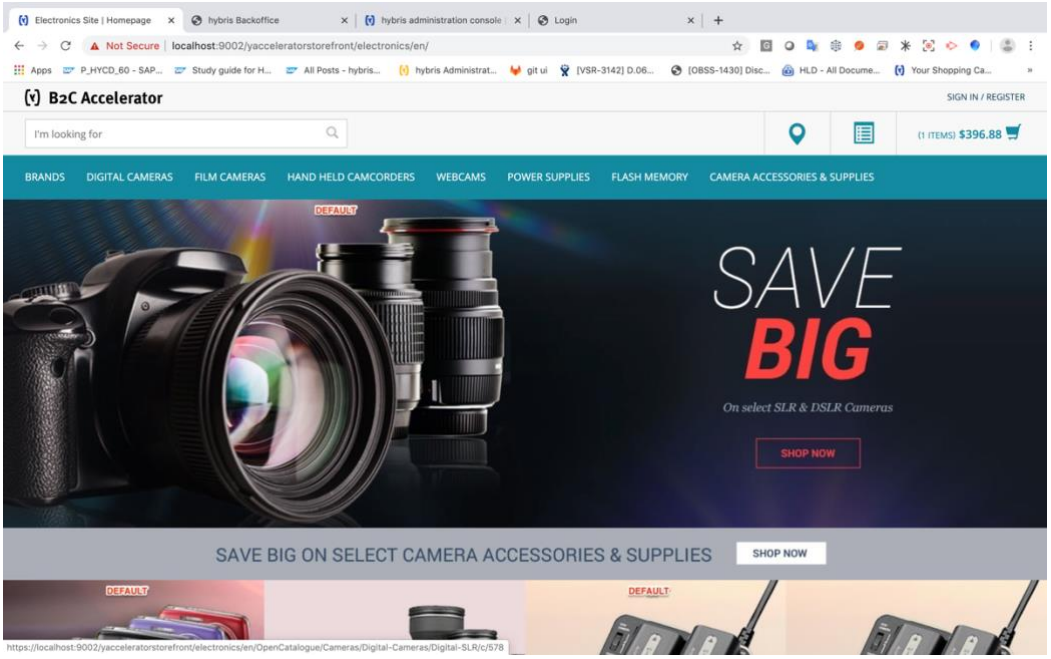


- **After double clicking the trigger window appears same as below, update the value of cron expression. Example given expression 0 0 12 * *? Means run cron job at 12:00 everyday. Example want to run cron job after every 30 minutes so expression will be updated as 0 30 0 * * ?**

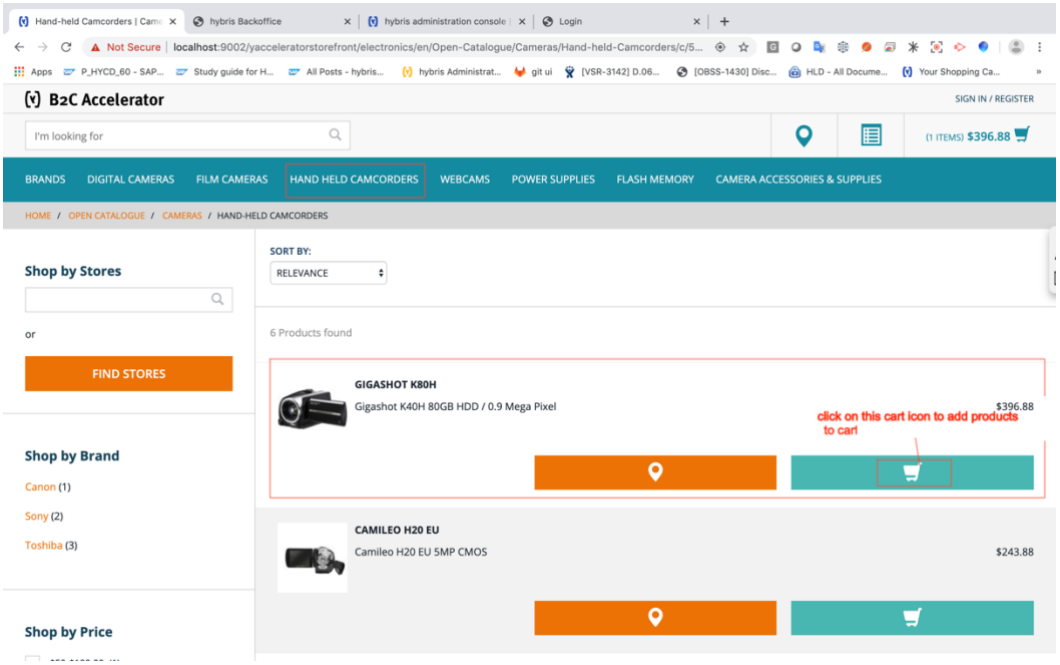


3.5.3 Testing of Plugin

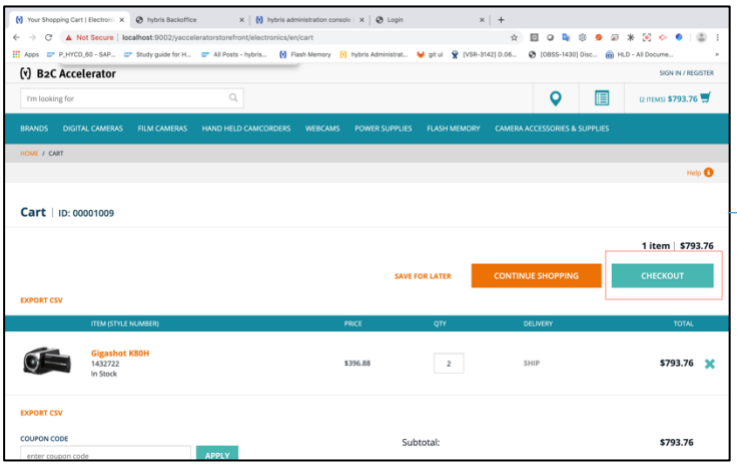
1. **Go to demo site storefront to start with placing order with Openpay. On browser add URL https://localhost:9002/yacceleratorstorefront/?site=electronics. The highlighted domain can be changed based on server. Taking an example we deploy code to staging environment and test it from any browser, then the URL becomes https://serverName:9002/yacceleratorstorefront/?site=electronics where serverName is name or id of server like ec1-23-45-216-100.us-west-2.compute.amazonaws.com**

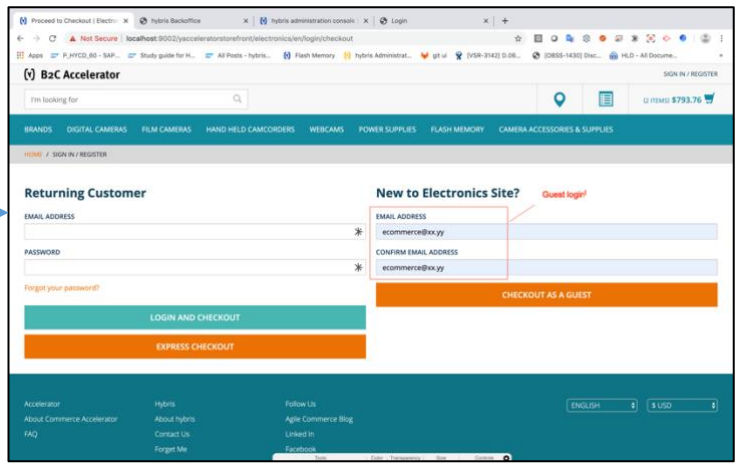


2. **Now select products and proceed to checkout**

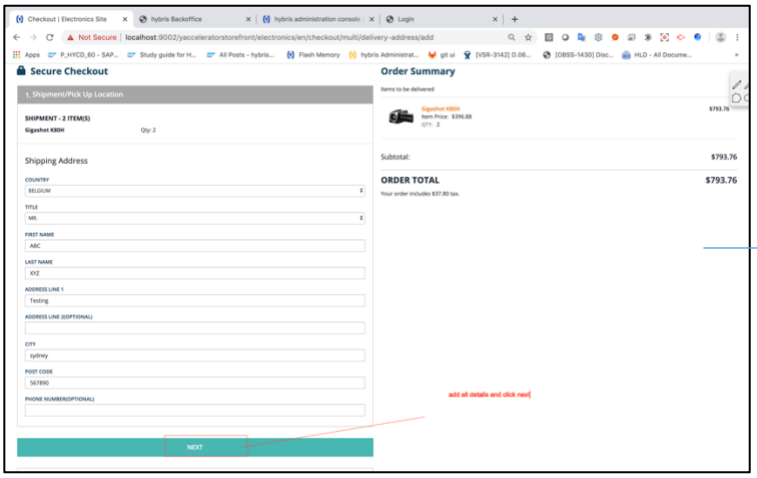


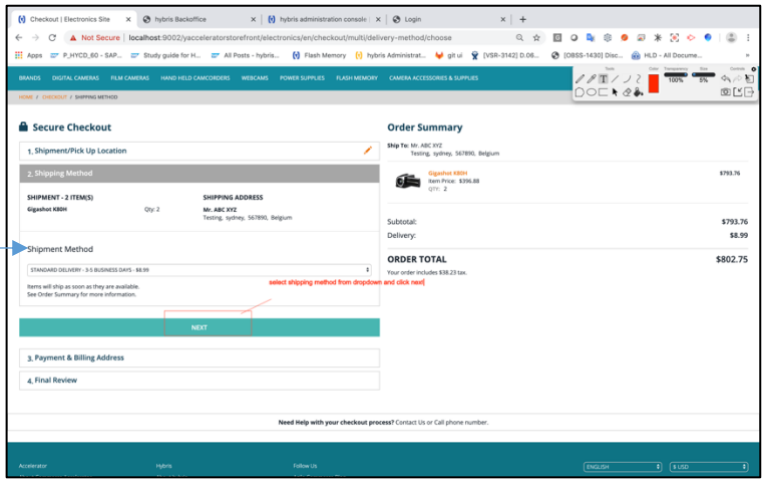
3. **Create a new user or login or checkout as guest login from checkout page. For testing use email ecommerce@xx.yy**



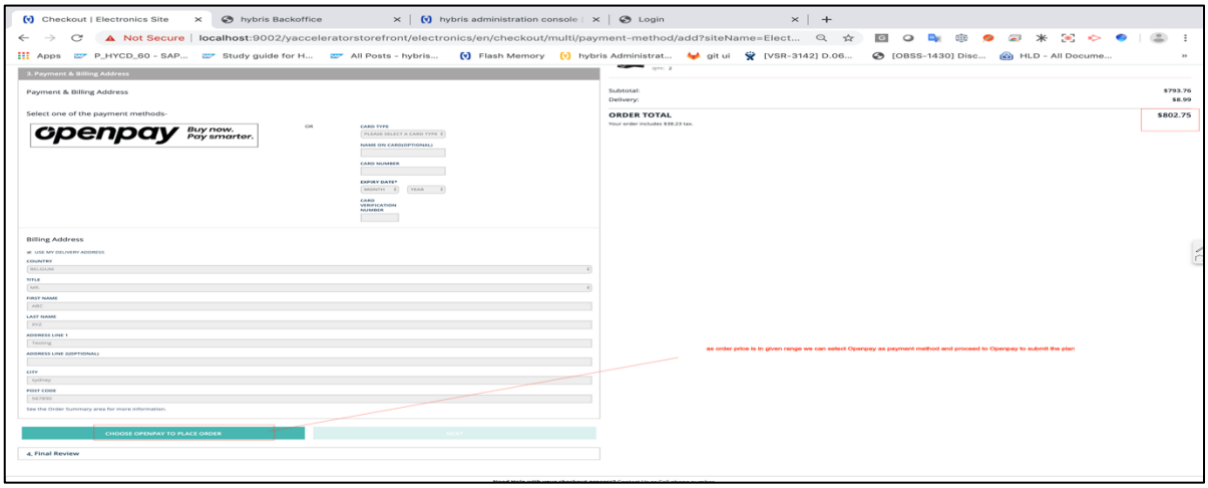


4. **After login successfully we move to checkout page to add following details**

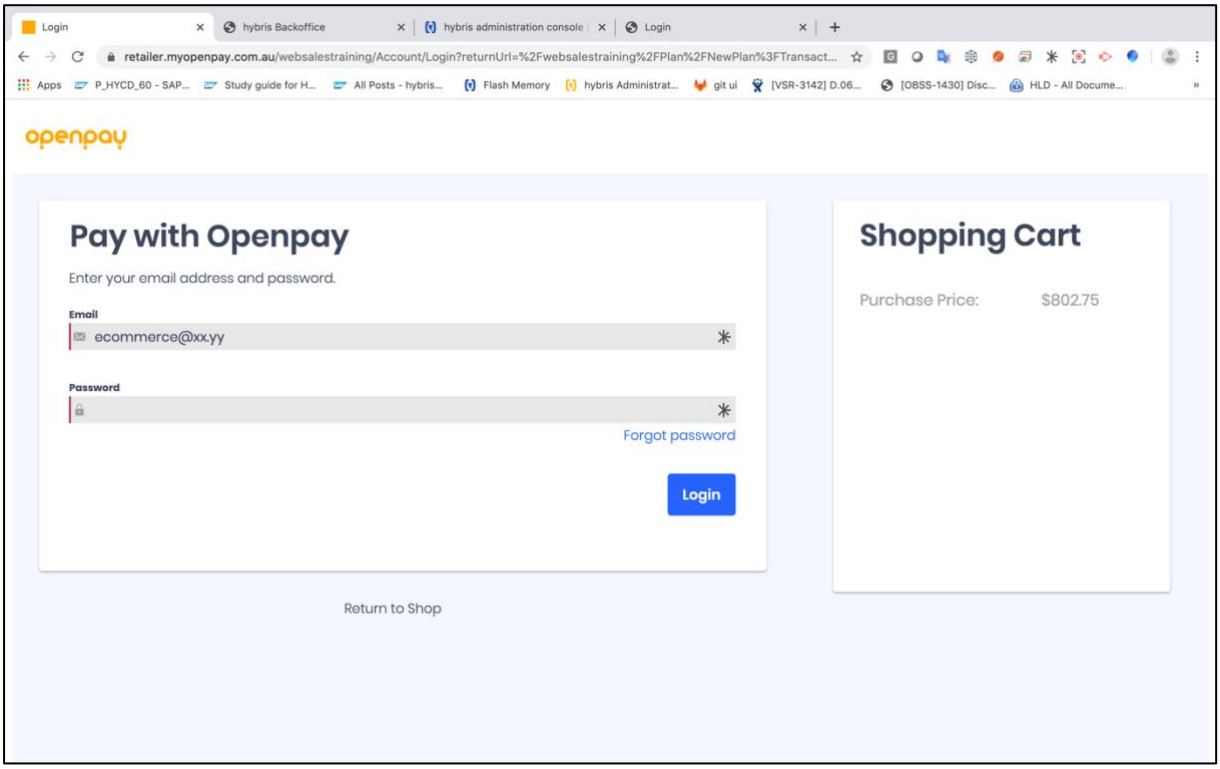




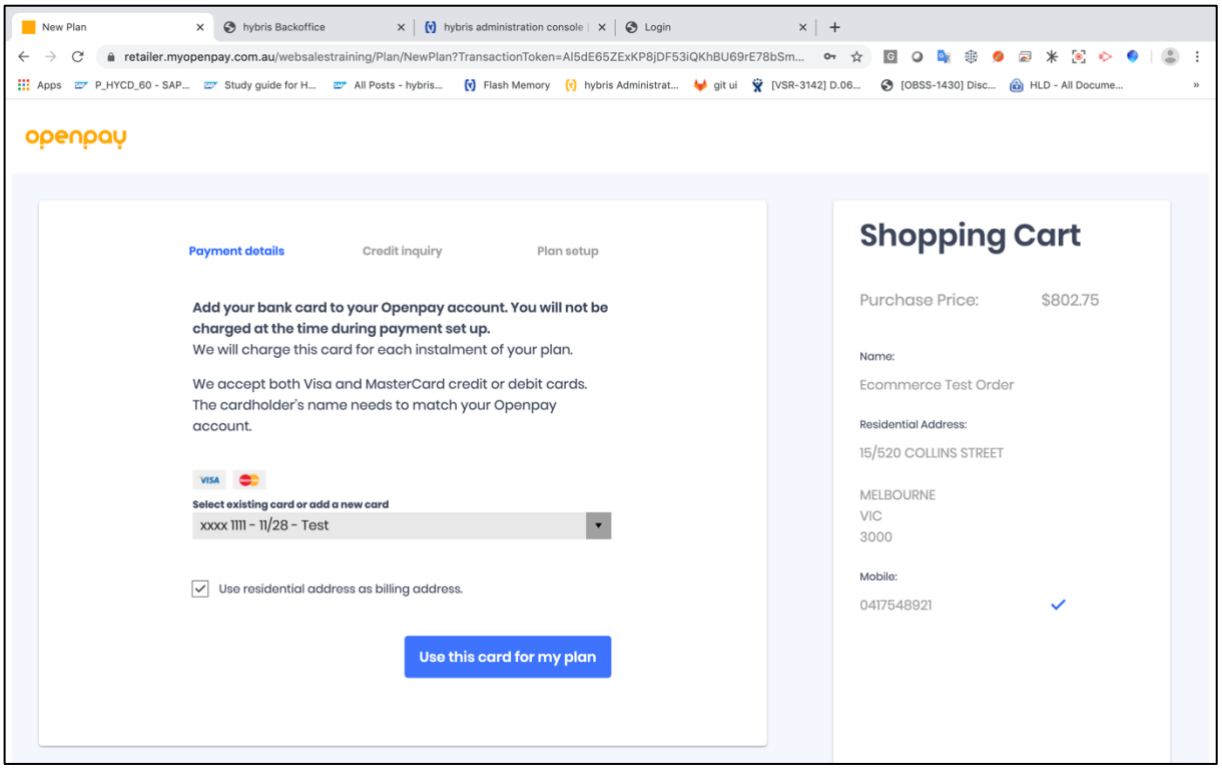
5. **After successfully adding shipping details we move to payment section where along with OOTB payment method Openpay payment method is added. This method is selected only if order price is between given range configured.**

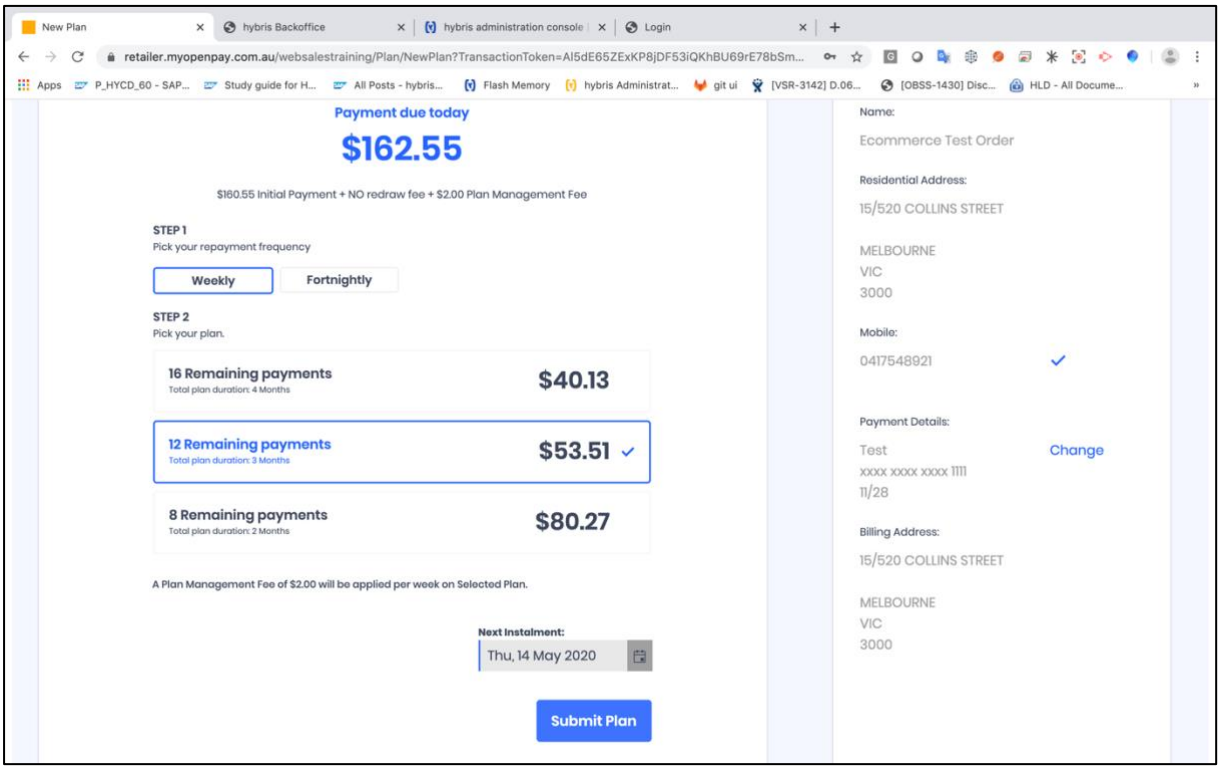


6. **On clicking "Choose Openpay to place order" create new order api is called from Openpay SDK and with transaction id and redirect URL we are redirected to Openpay to select and submit the plan. Use login credentials as ecommerce@xx.yy/Test@1001 for testing. Check logs in Hybris console terminal side by side**

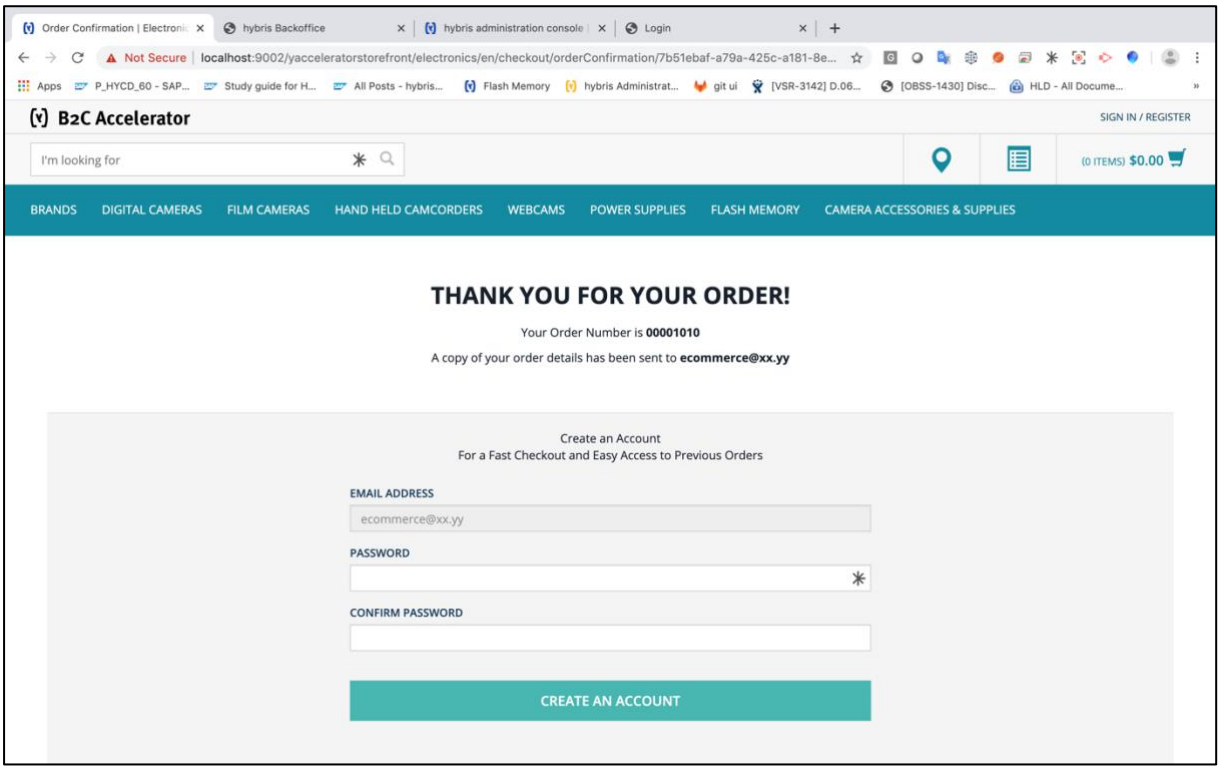


7. After login to Openpay perform further steps to submit the plan by giving card details and choosing the plan.

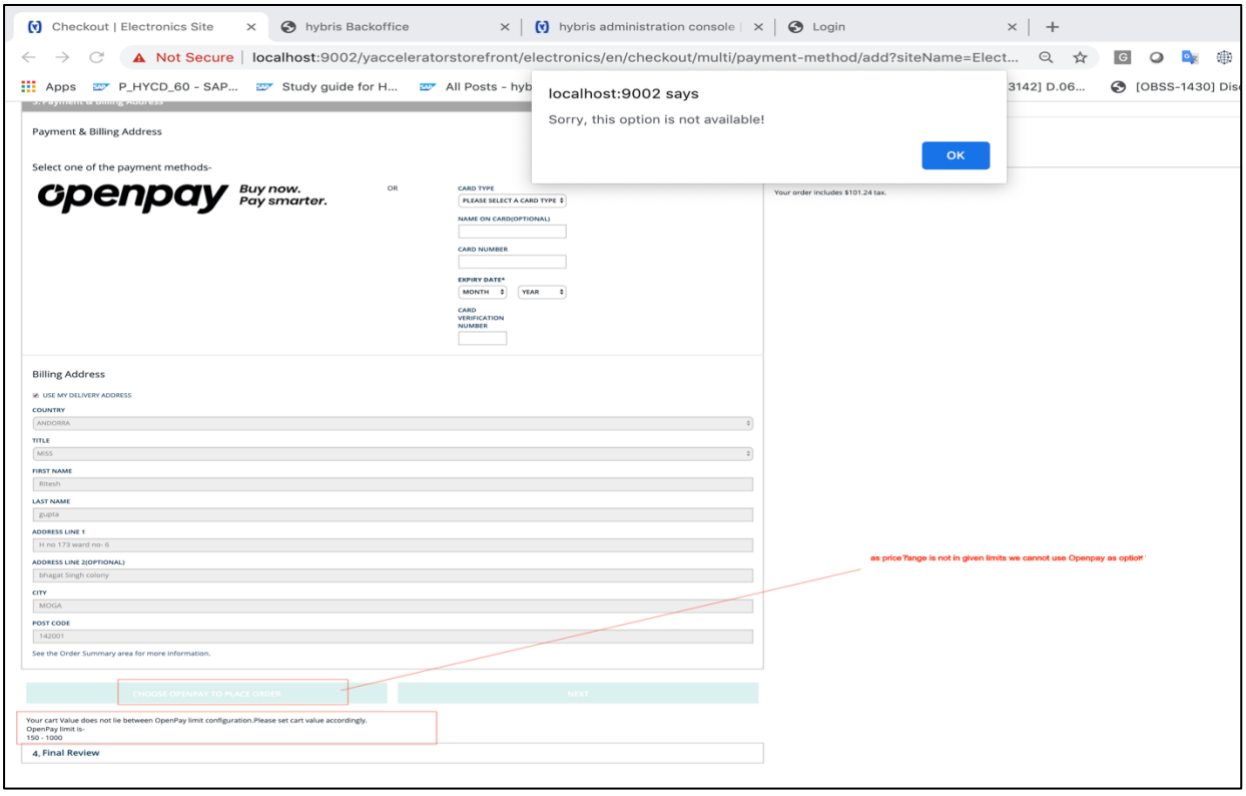




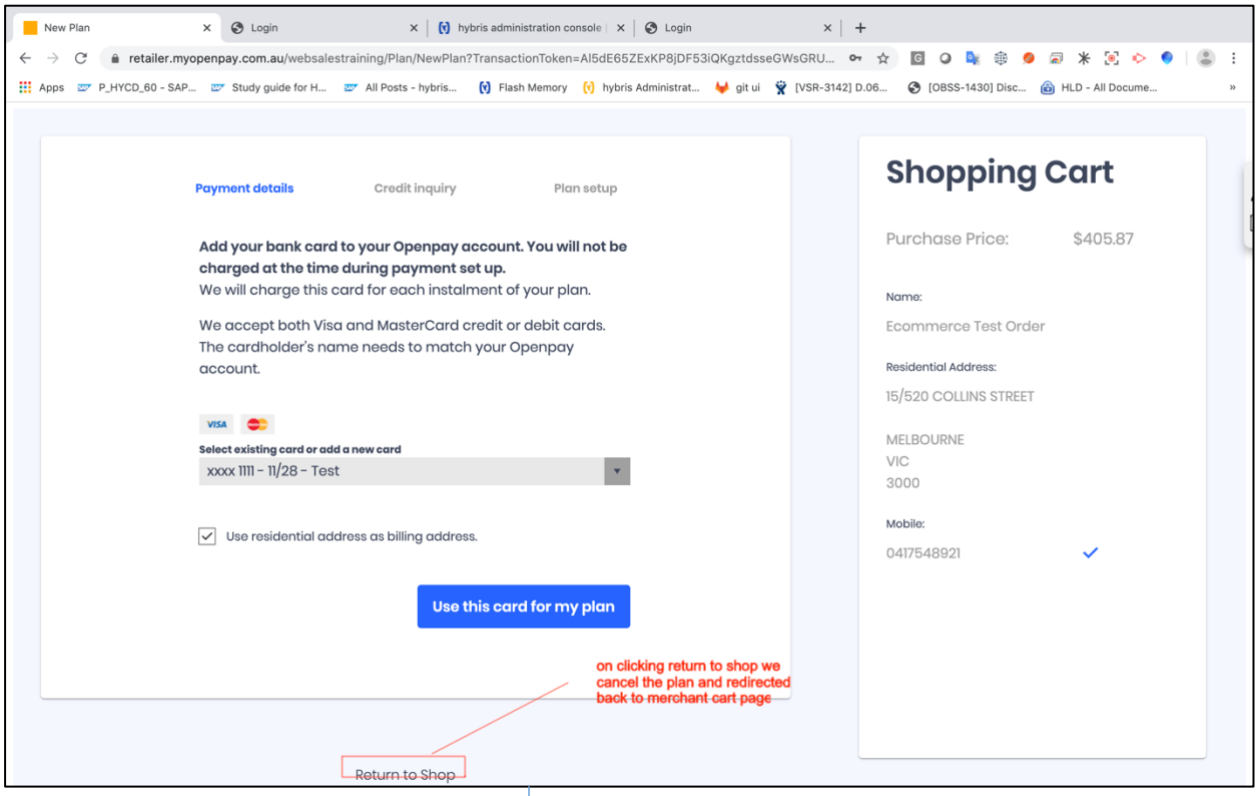
8. **After submitting the plan successfully, order is created in Hybris and we land to order confirmation page.**

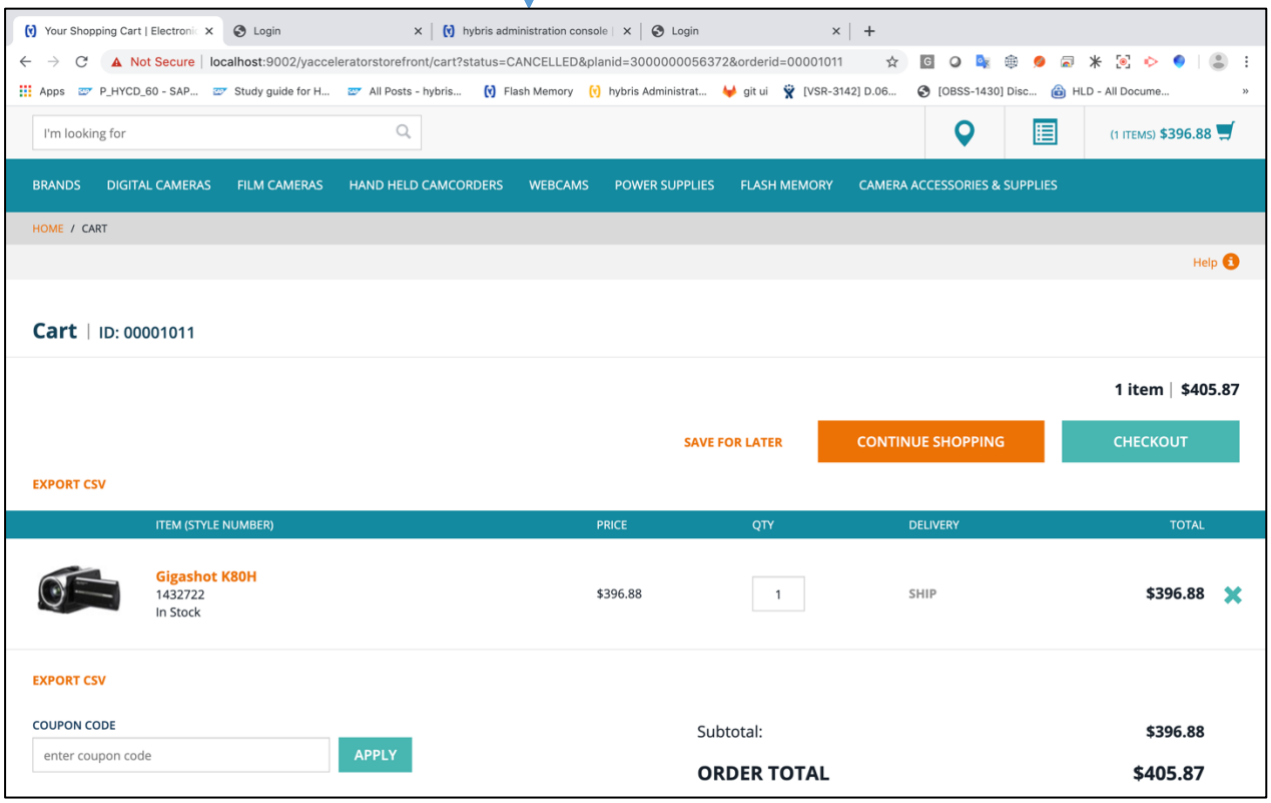


9. **If cart price is not in range of configured values we cannot move further with payment option as Openpay**

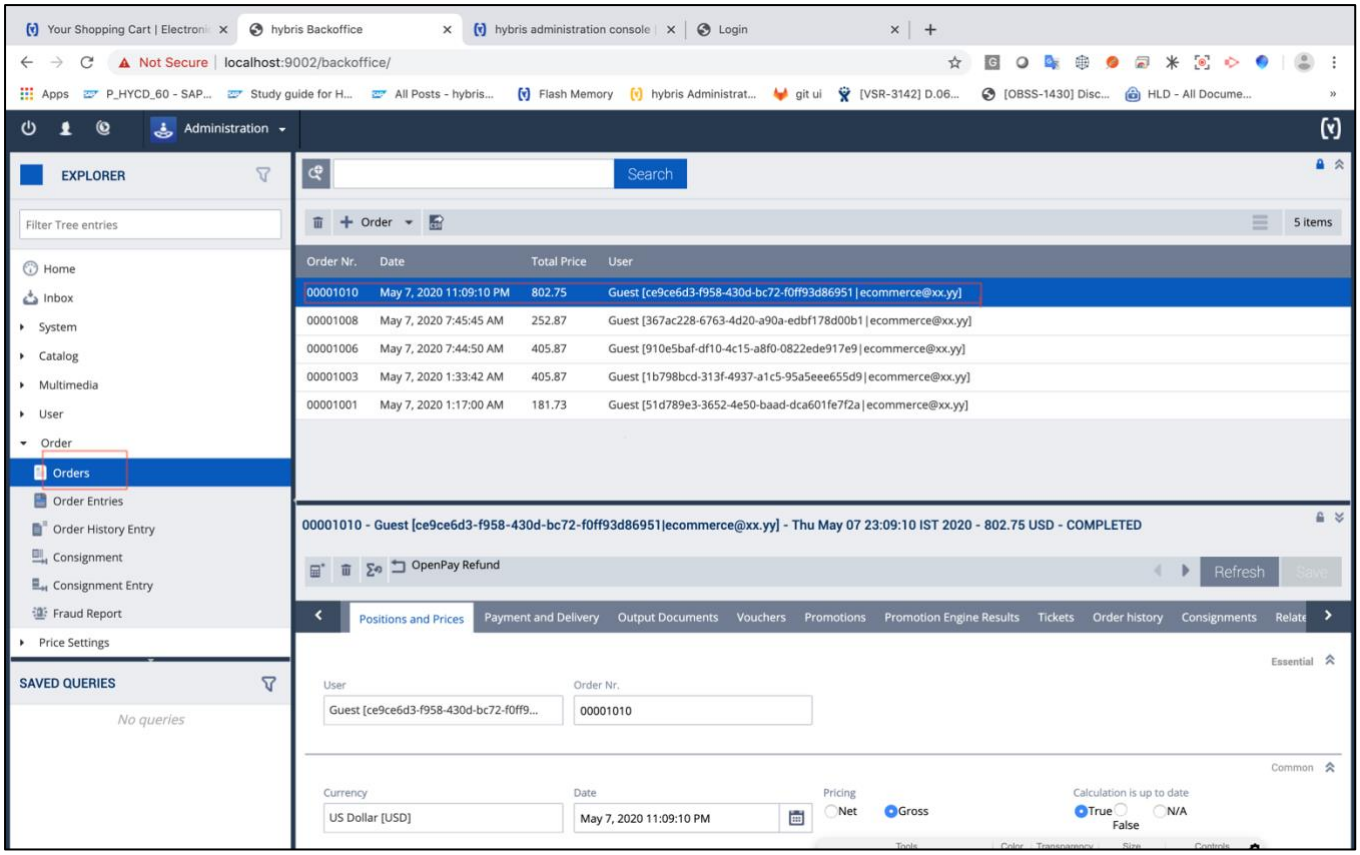


10. **If we are redirected to Openpay but instead of submitting order we select “return to shop” or if browser is interrupted or closed abruptly order is not created in Hybris and we land on cart page.**

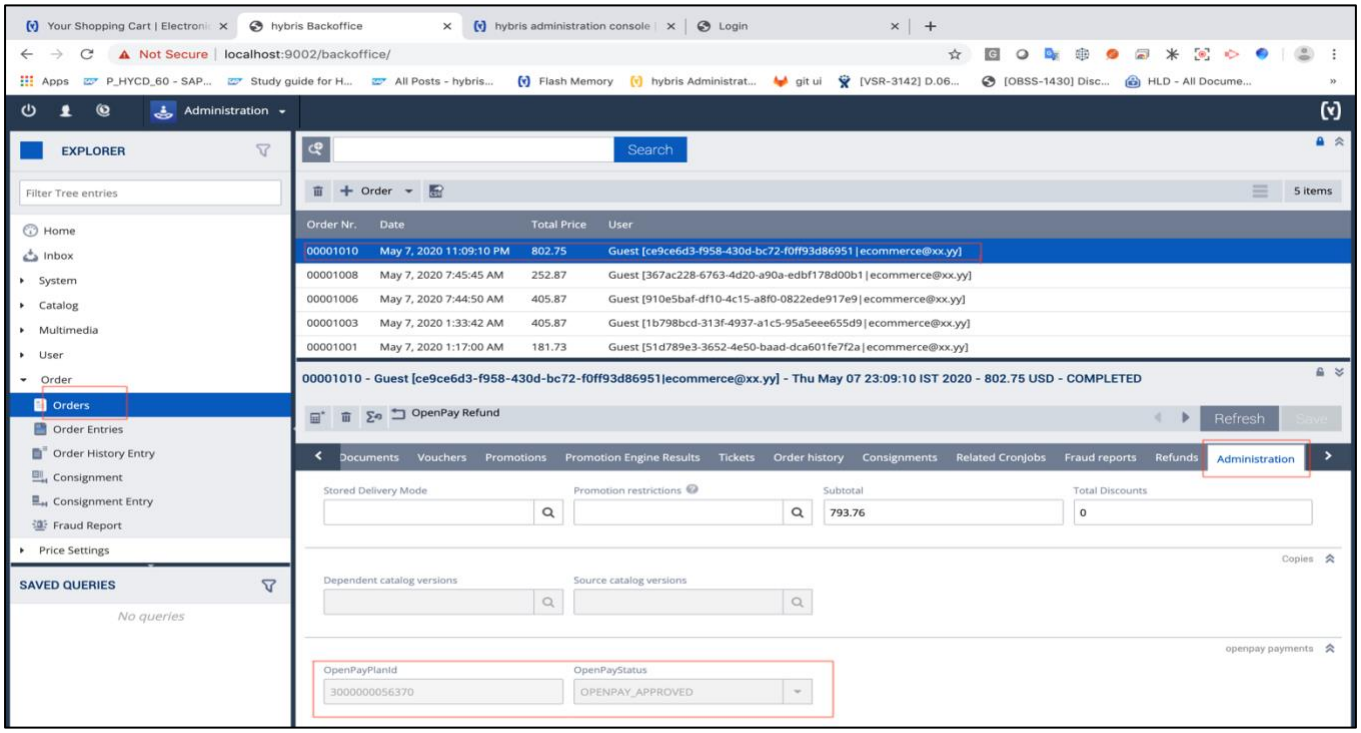




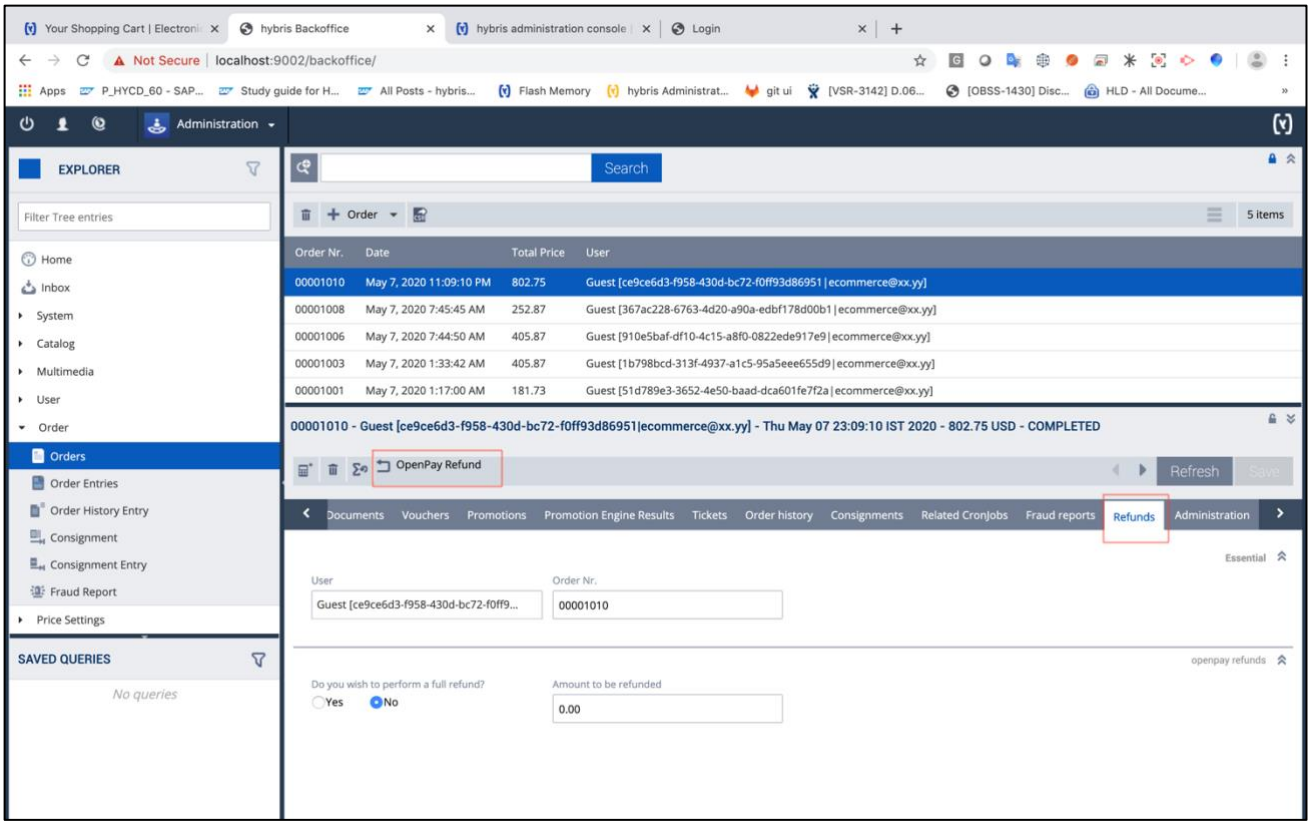
11. **Once order is placed successfully in Hybris we can check order in backoffice and we can perform refunds if possible. Just in above screenshots check we created order with order id "00001010", check order confirmation page. We check the order with same id on backoffice.**
Login to backoffice (https://localhost:9002/backoffice) --> Login with credentials (admin/nimda) → Go To Order tab



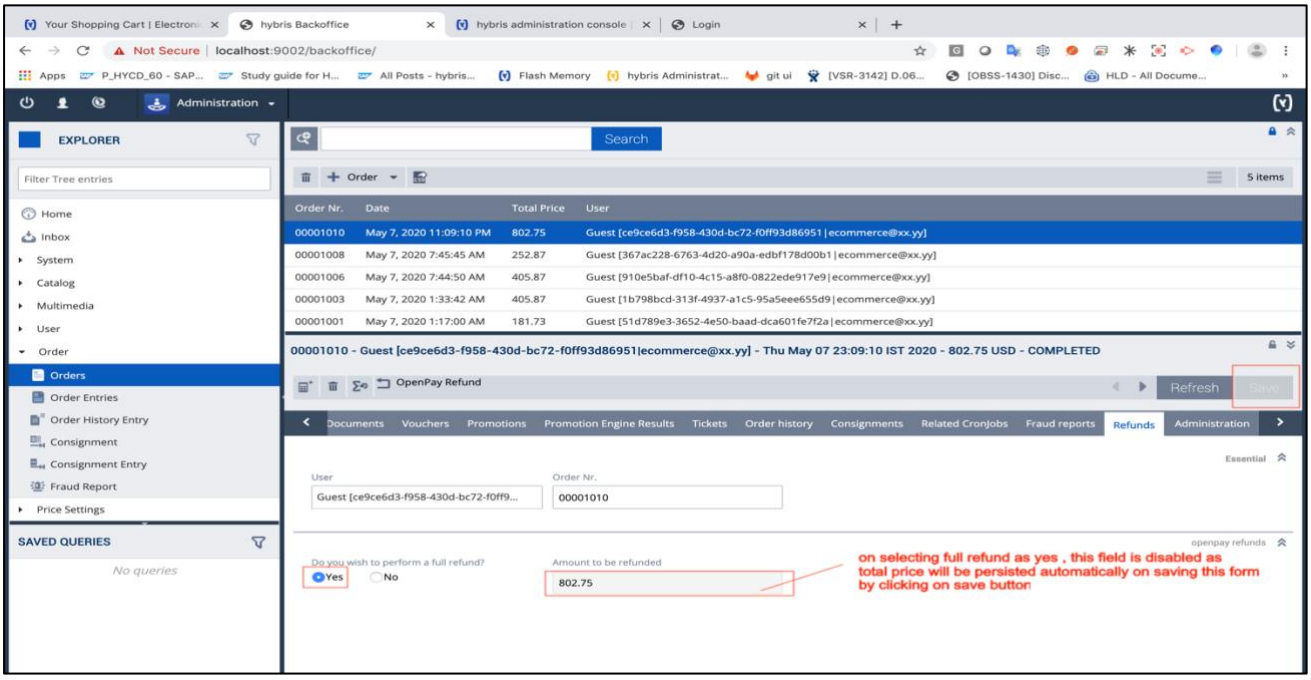
12. **When we placed order with Openpay, order status and plan id is stored in DB but not editable.**



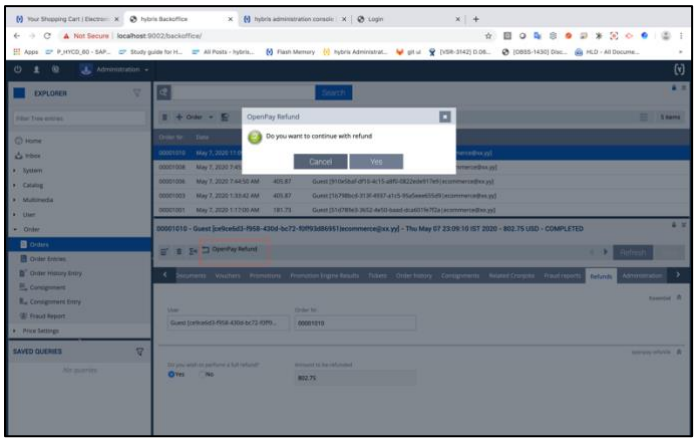
13. **For refunds, order should be shipped or pickup complete (consignment should be created) and Openpay order status should be OPENPAY_APPROVED to activate refunds. Refund button is available on editor area of tab as shown in screenshot above.**

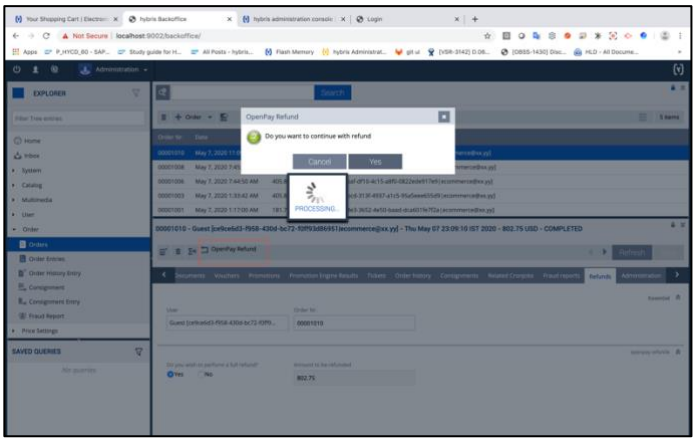


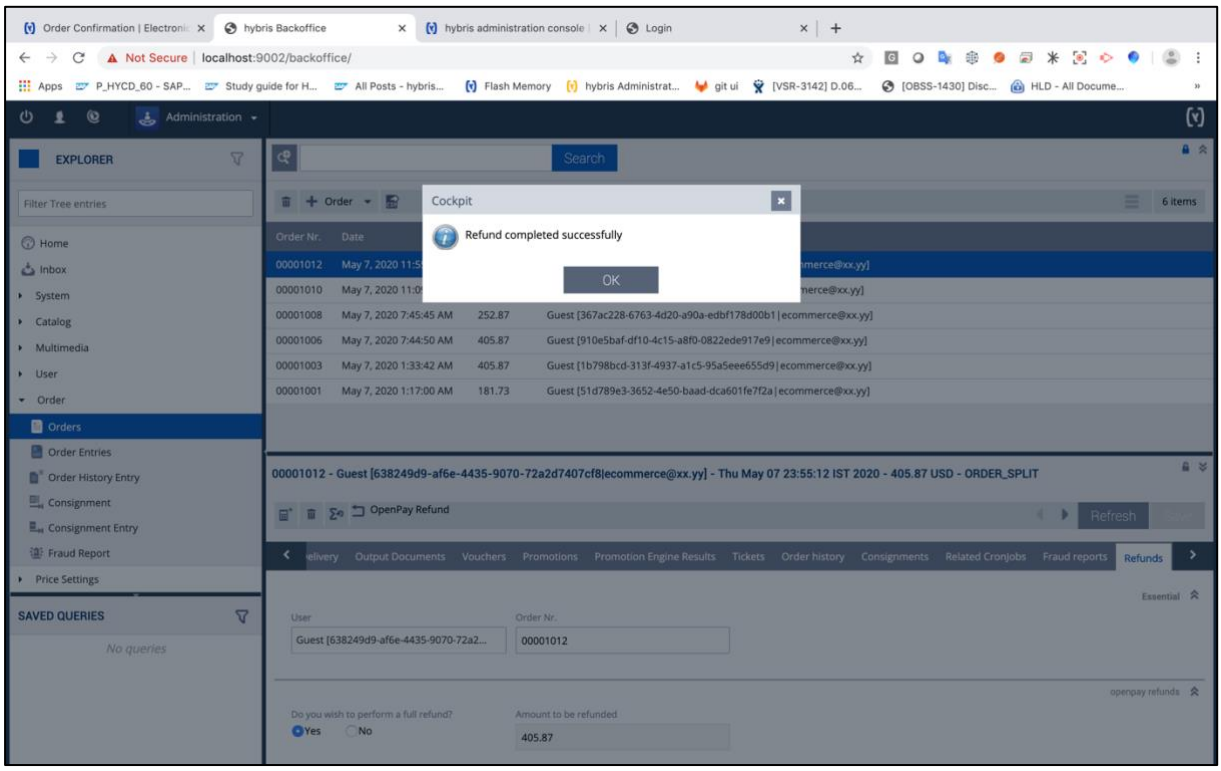
14. **Before clicking on refund button add details on refund tab like whether its full or partial refund, amount to be refunded. If its full refund just select Yes and save the form, value will be persisted automatically. For partial refund select No and add amount to be refunded.**



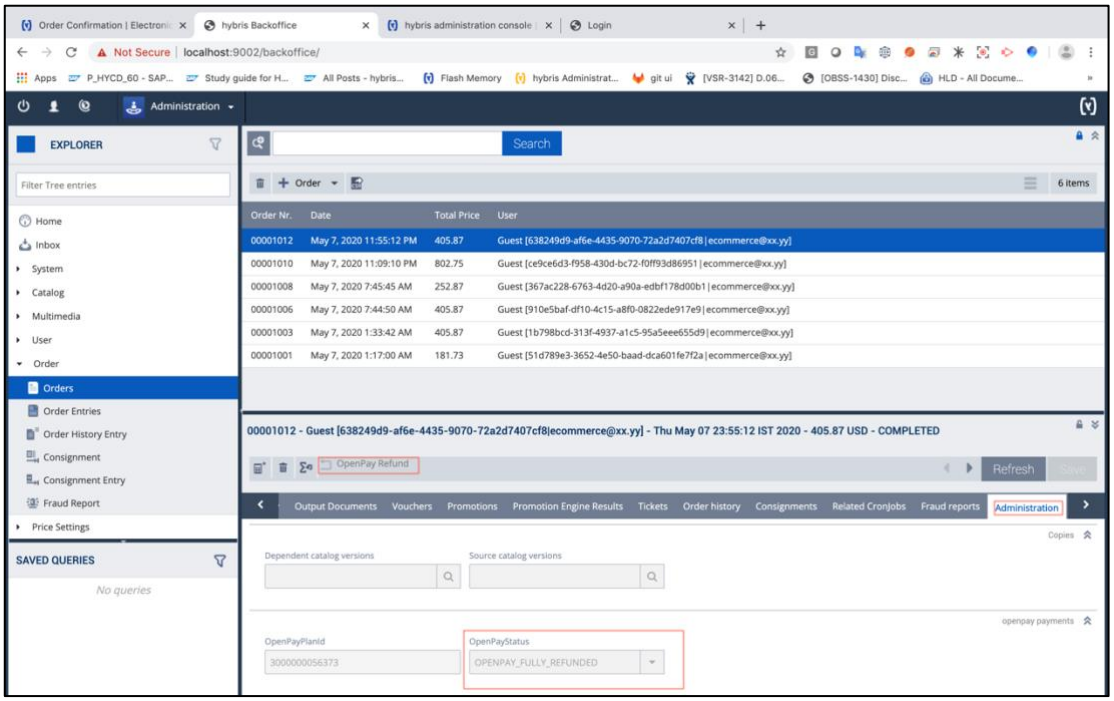
15. **Now click on refund button and on confirmation, refund is successfully completed if data is correctly sent.**



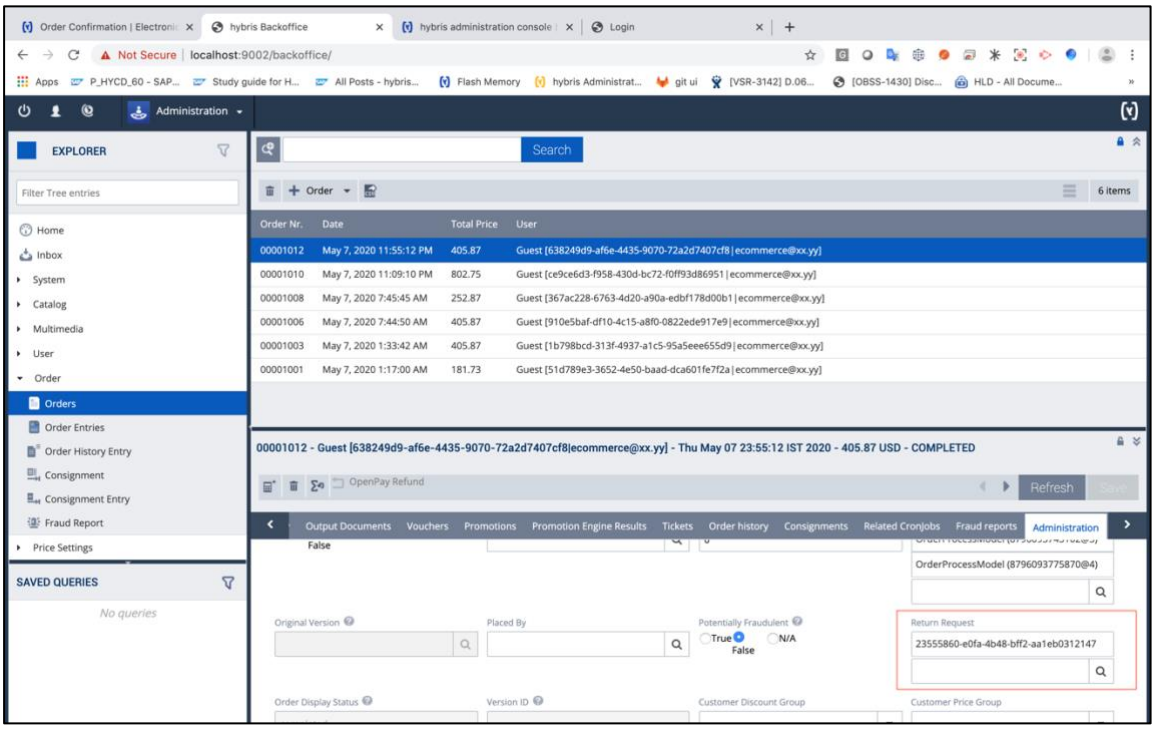




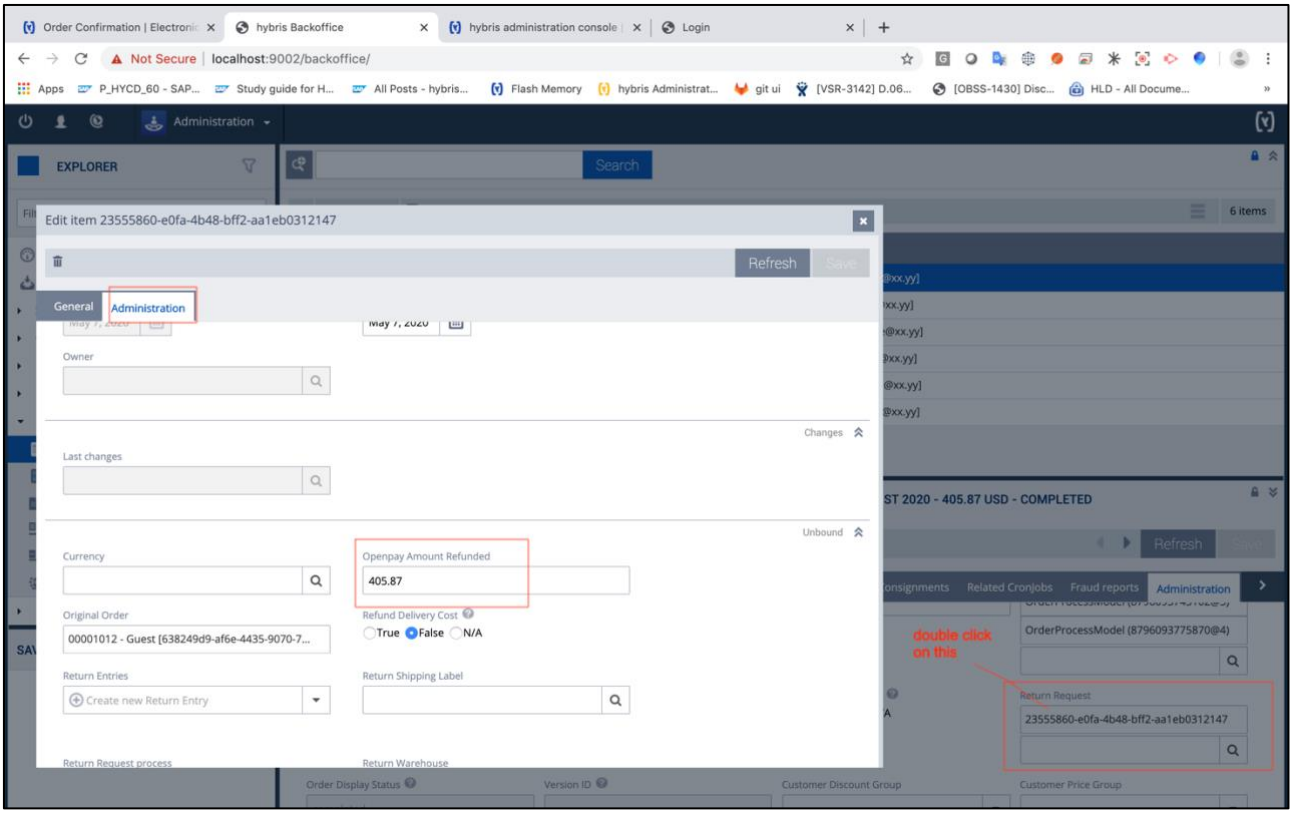
16. **Once order is fully refunded, refund tab is hidden and order status is updated to OPENPAY_FULLY_REFUNDED. Refund button is also disabled.**



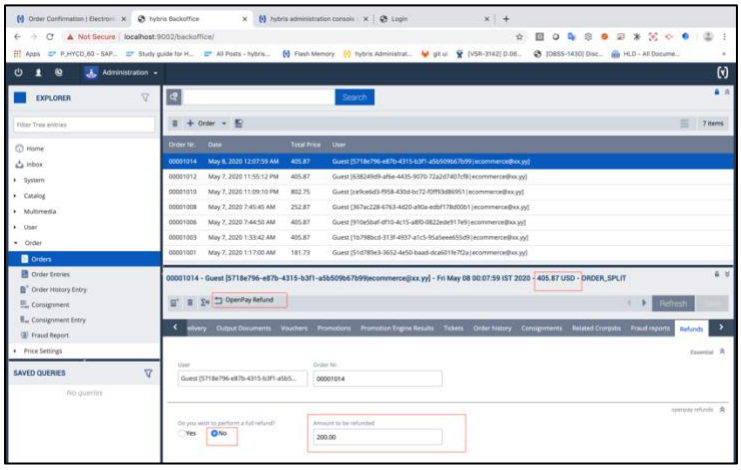
17. **We can check the return history from same tab under return process attribute. If its full refund one entry is added, for partial refunds multiple entries are created.**

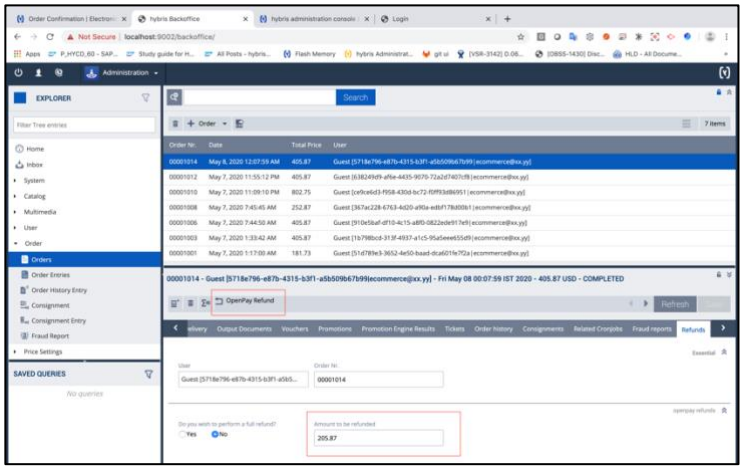


18. **Double click on the above return request entry to check refund details**

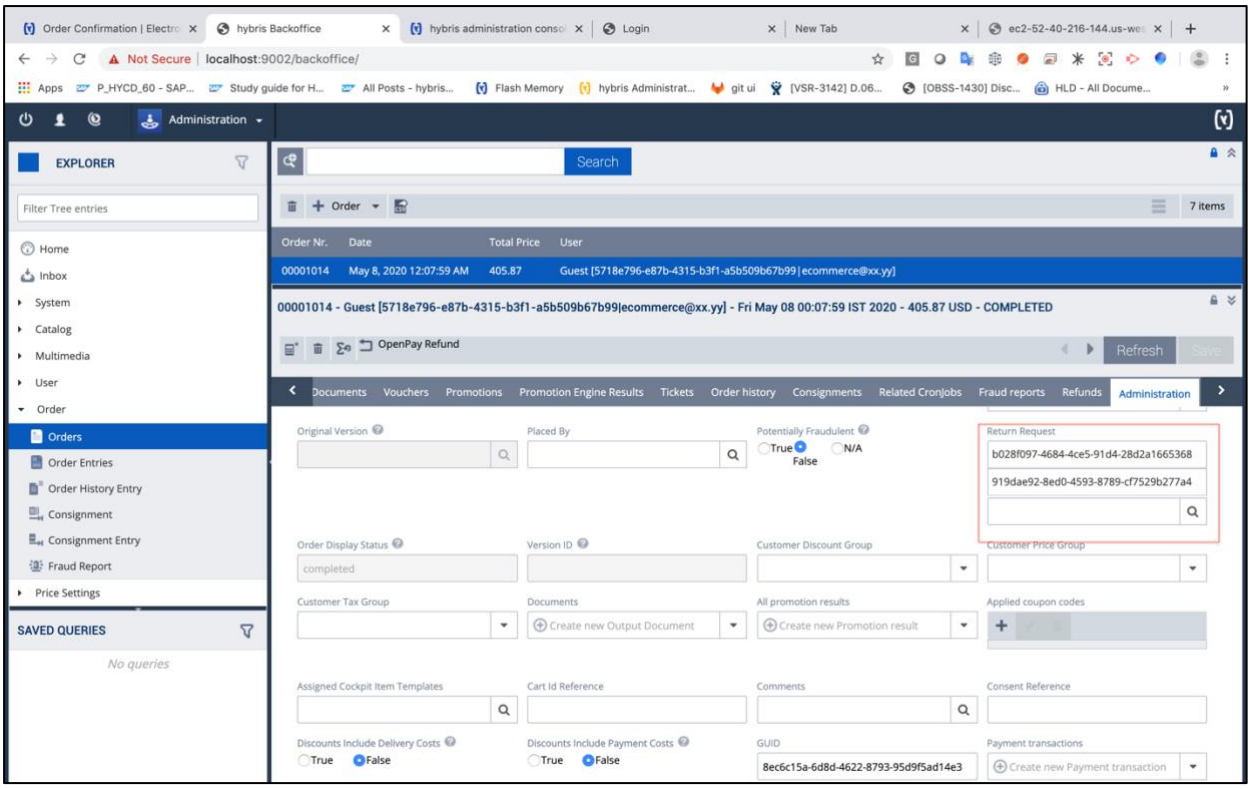


19. **For partial refunds, we can do multiple partial refunds, everytime amount to be refunded is calculated again and updated.**



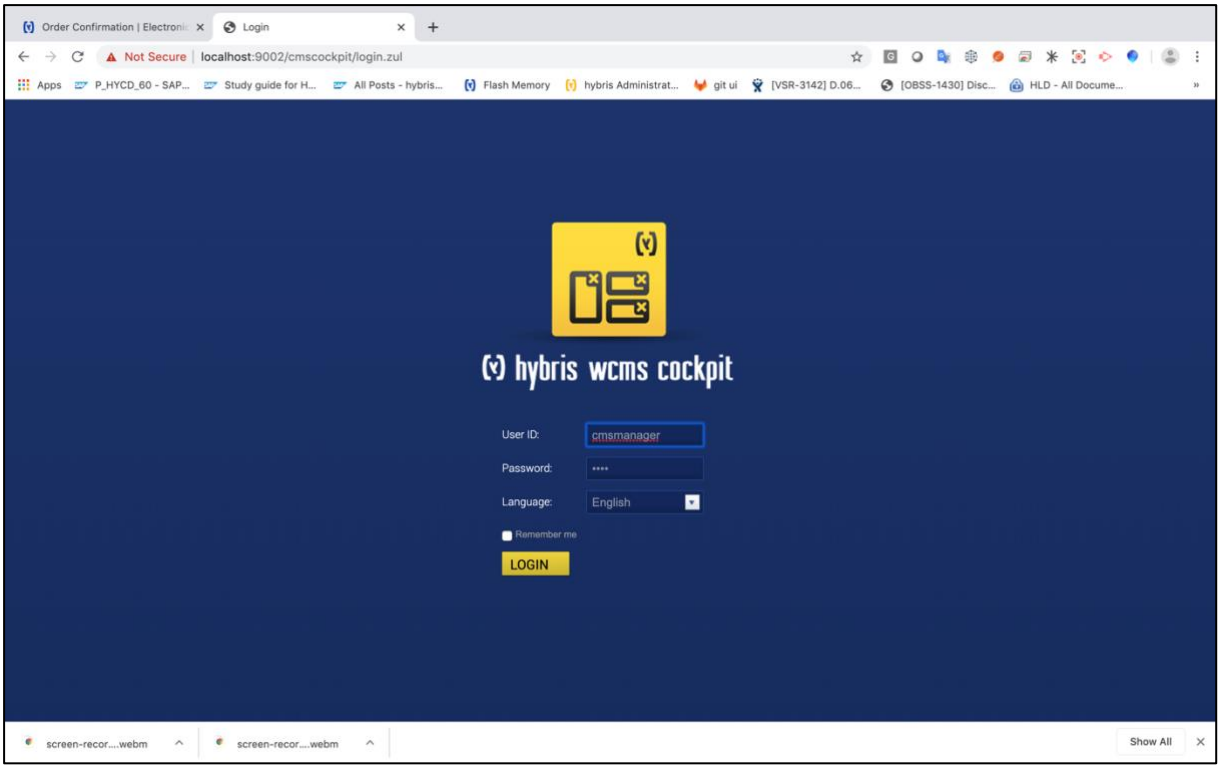


20. **We can perform multiple partial refunds by updating the value and history we can check through return request attribute.**

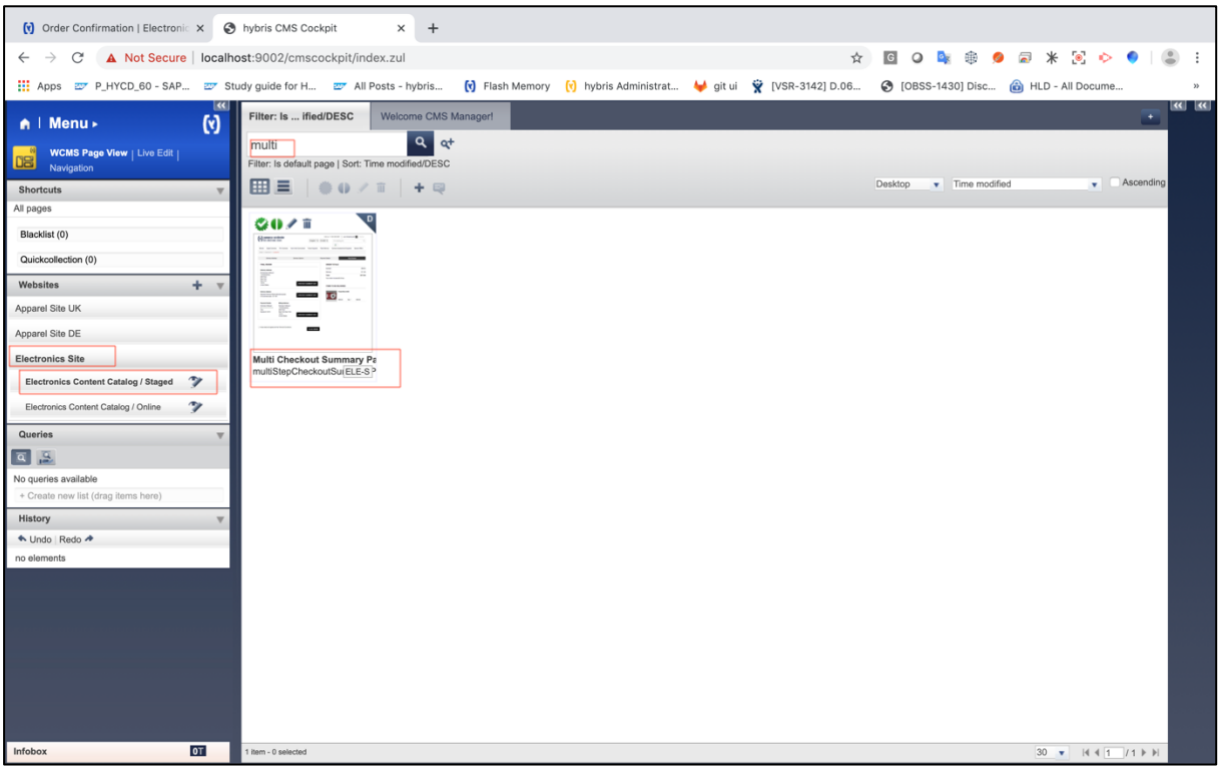


3.5.4 Configuration of Openpay Icon and text for storefront using cmscockpit

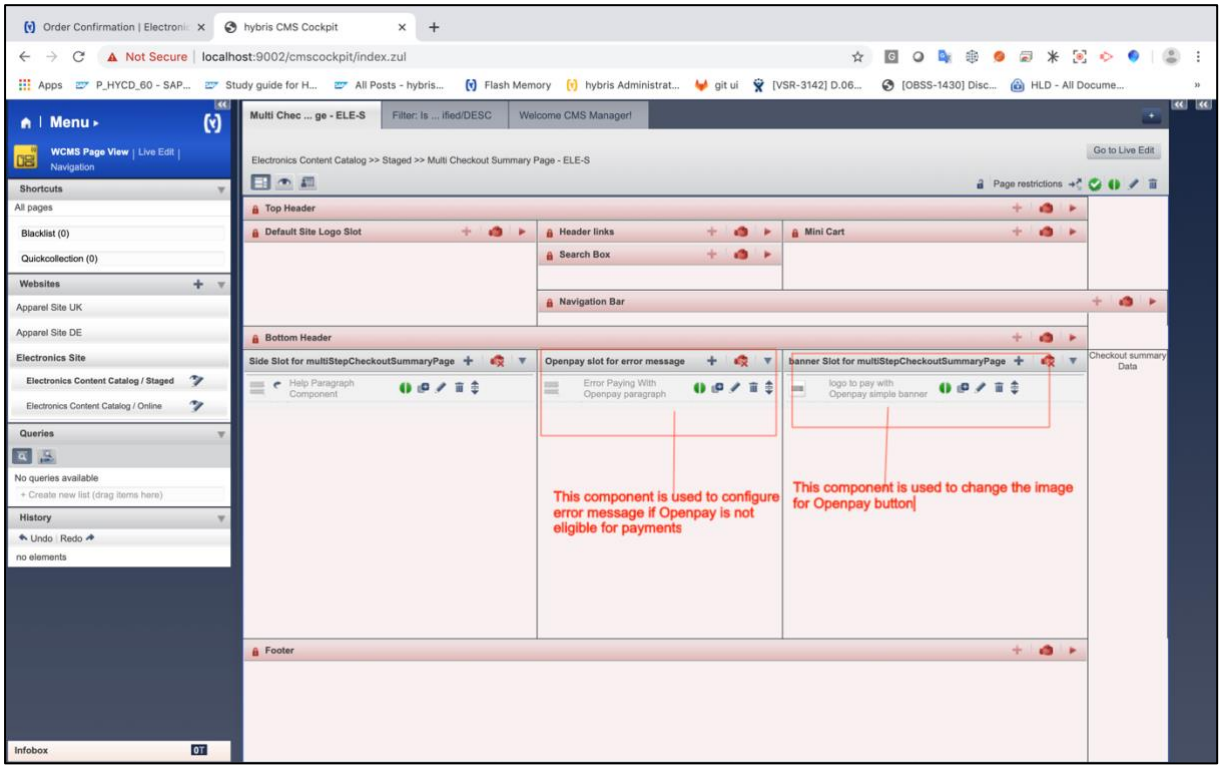
1. **We can configure Openpay icon by browsing to cmscockpit. https://localhost:9002/cmscockpit**



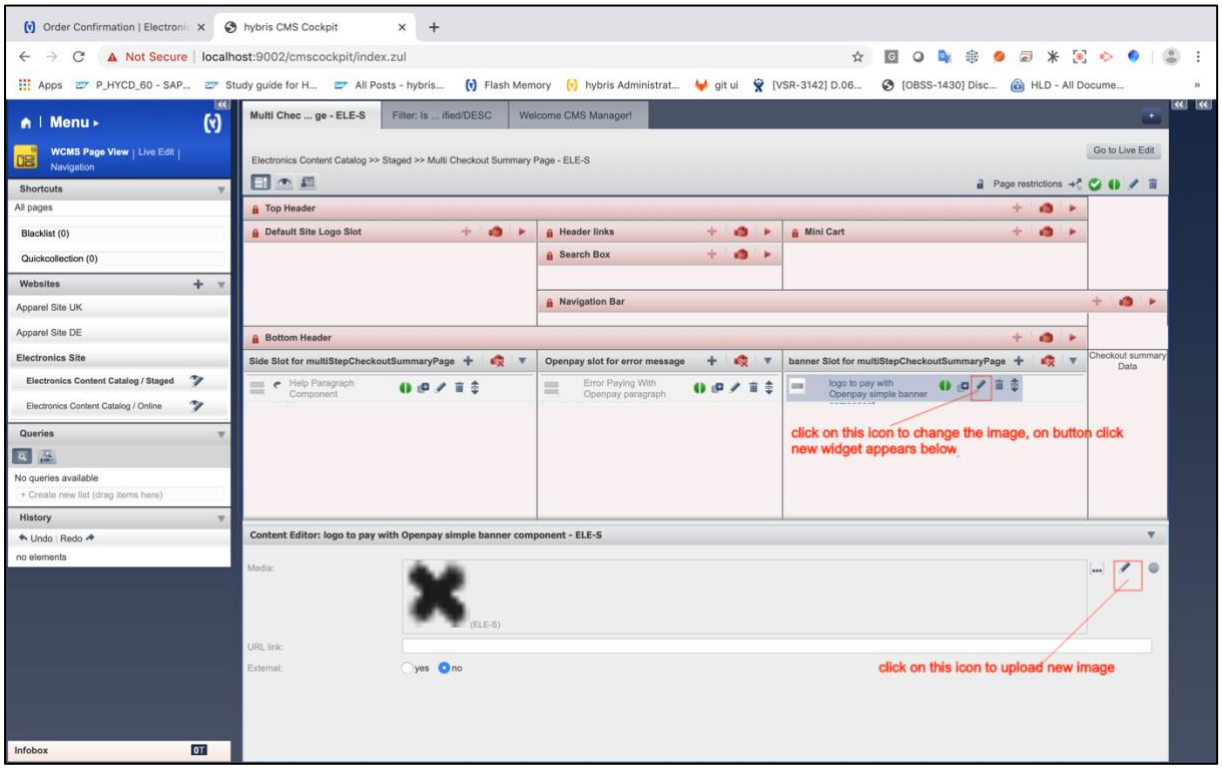
2. **Just select login as credentials are added automatically. Based on which store we are using for online shopping we select the catalog with staged version, changes will be done in staged version and then catalog will be synchronized. Select multi checkout summary page for particular catalog.**



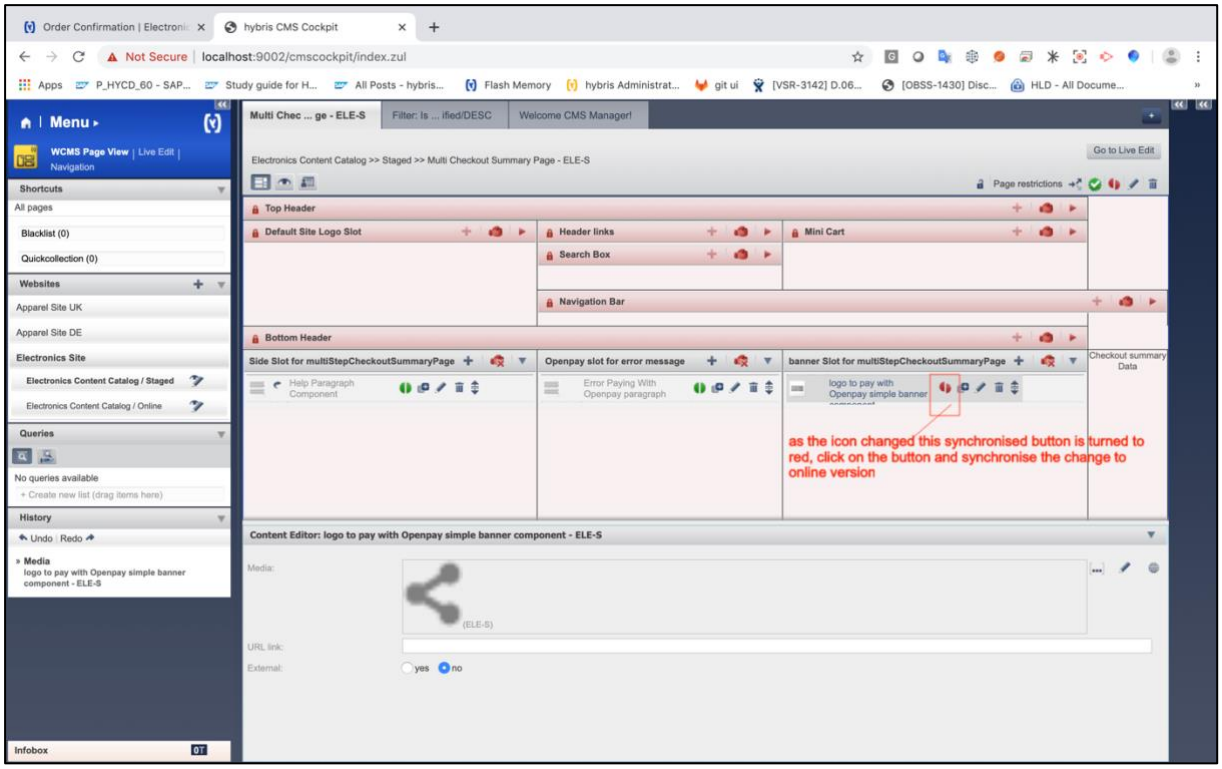
3. **Double click on page or click the edit icon for configuration of components.**



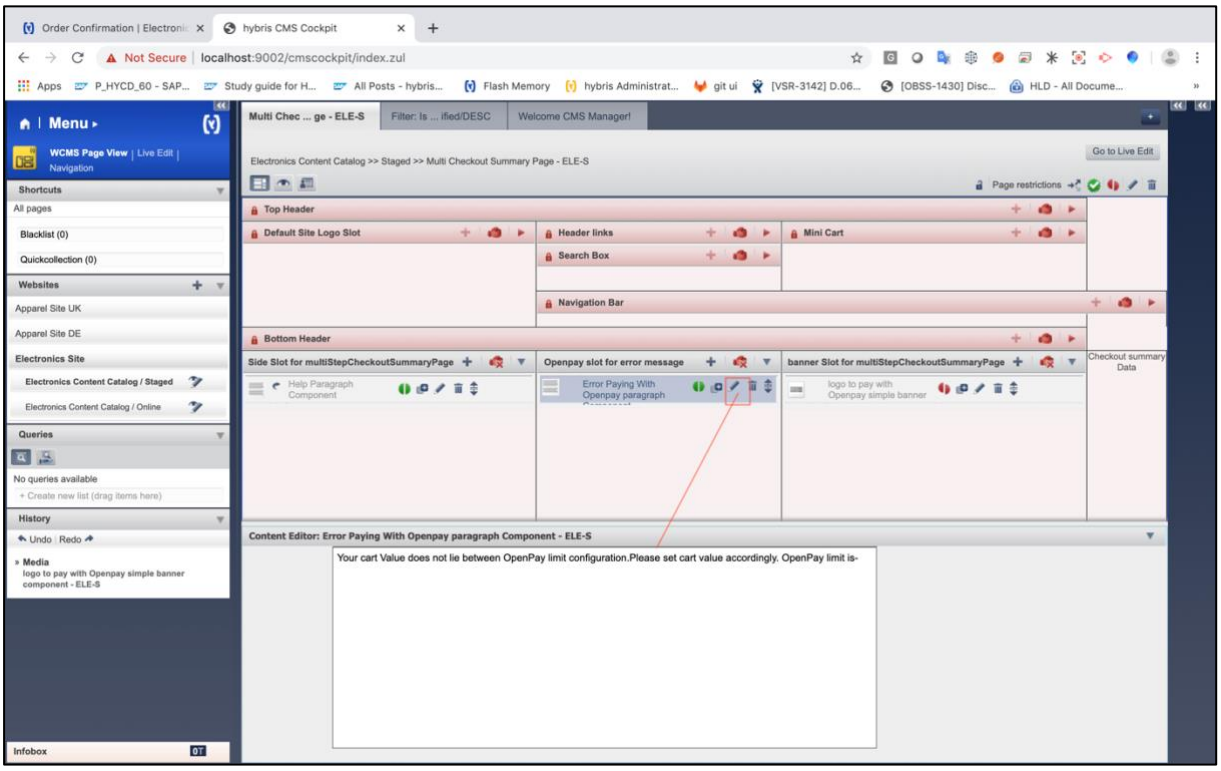
4. **Select Logo component to change the logo**



5. **As the logo changes the component synchronise button changes to red from green, just click on the button to render changes to frontend.**



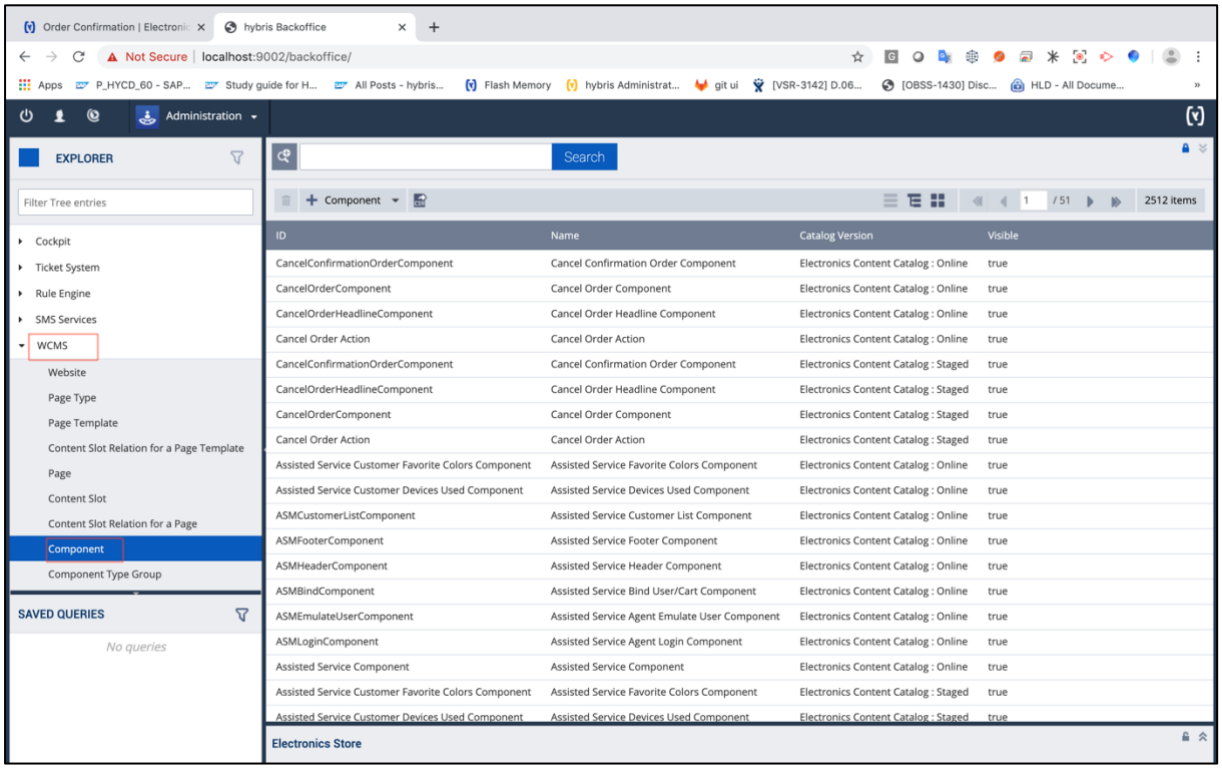
6. **Now in similar fashion text can be changed for error component.**



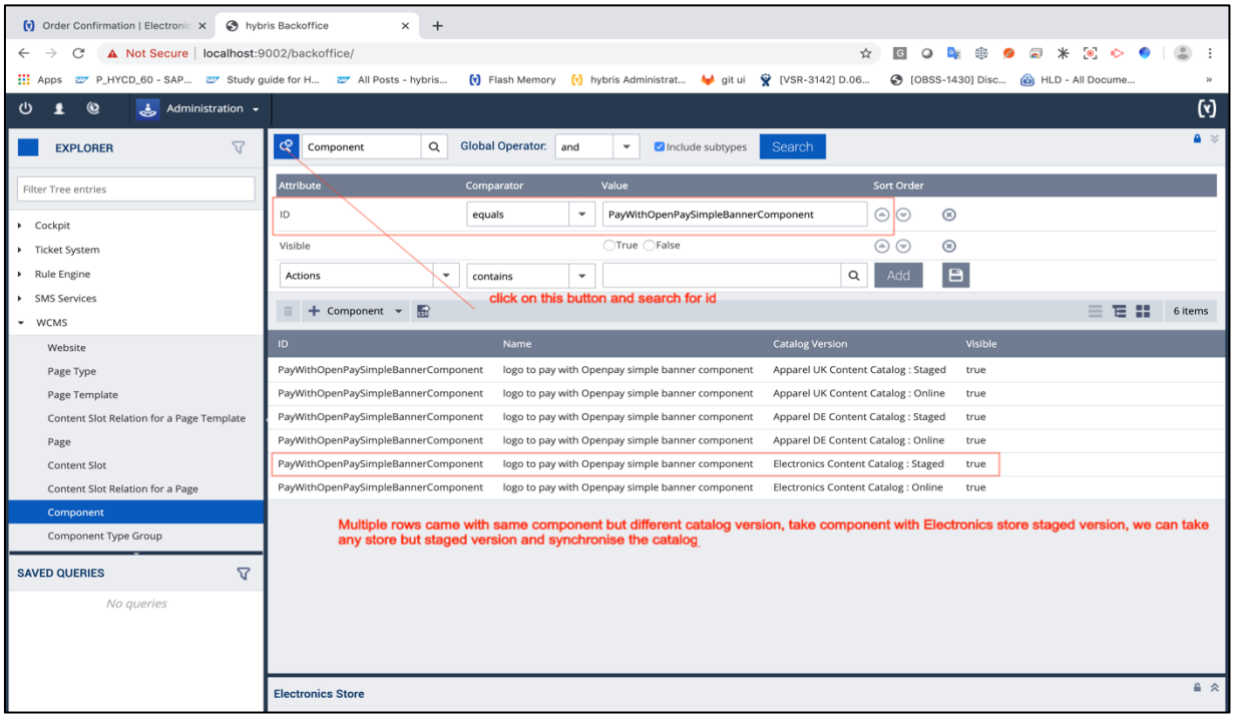
3.5.5 Configuration of Openpay Icon and text for storefront using backoffice

For some new Hybris versions cmscockpit is not used, so we can do above configurations using backoffice

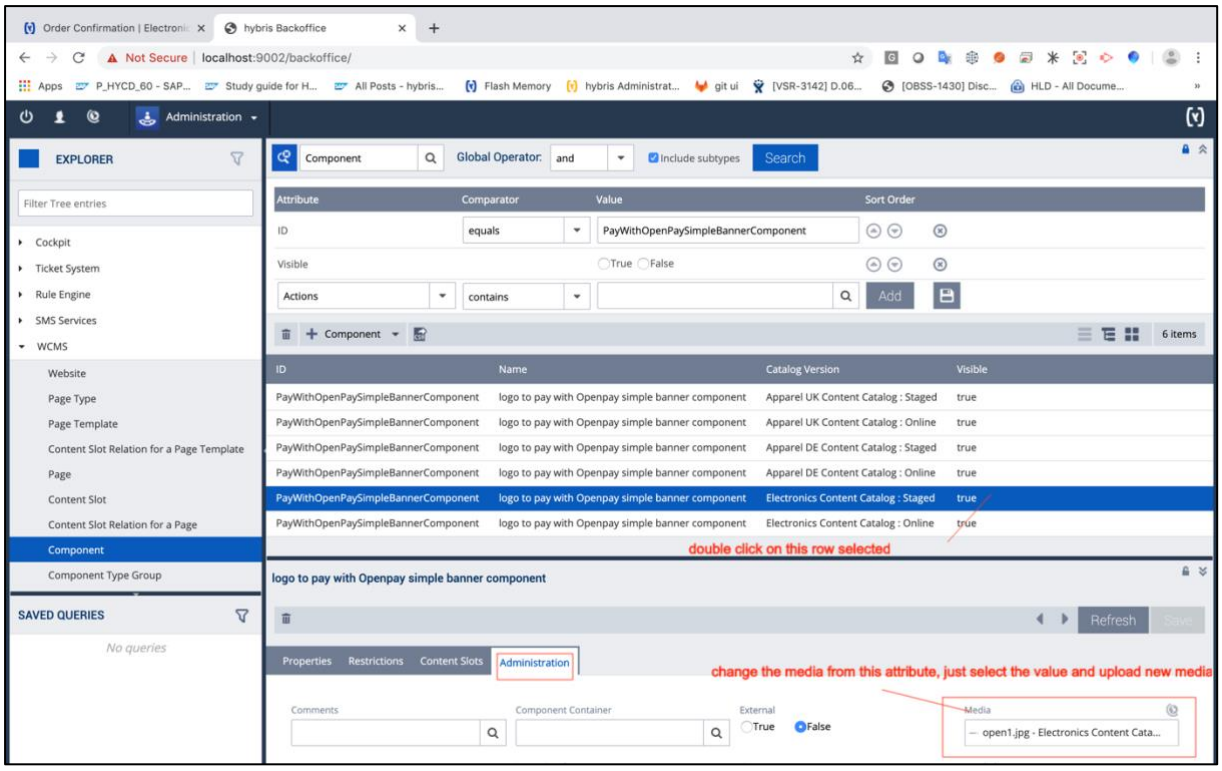
1. **Logon to backoffice and go to tab WCMS → Components**



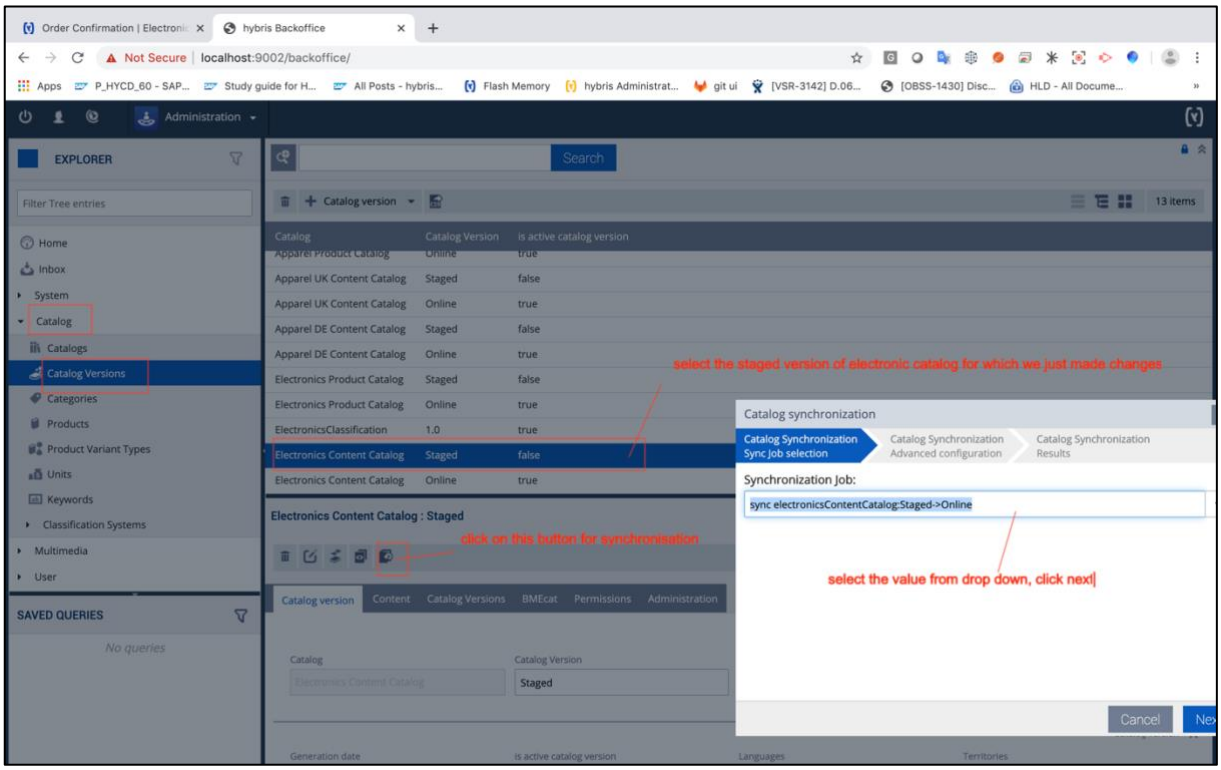
2. **We have created components by id ErrorPayingWithOpenPayParagraphComponent and PayWithOpenPaySimpleBannerComponent. Search these components in search box**

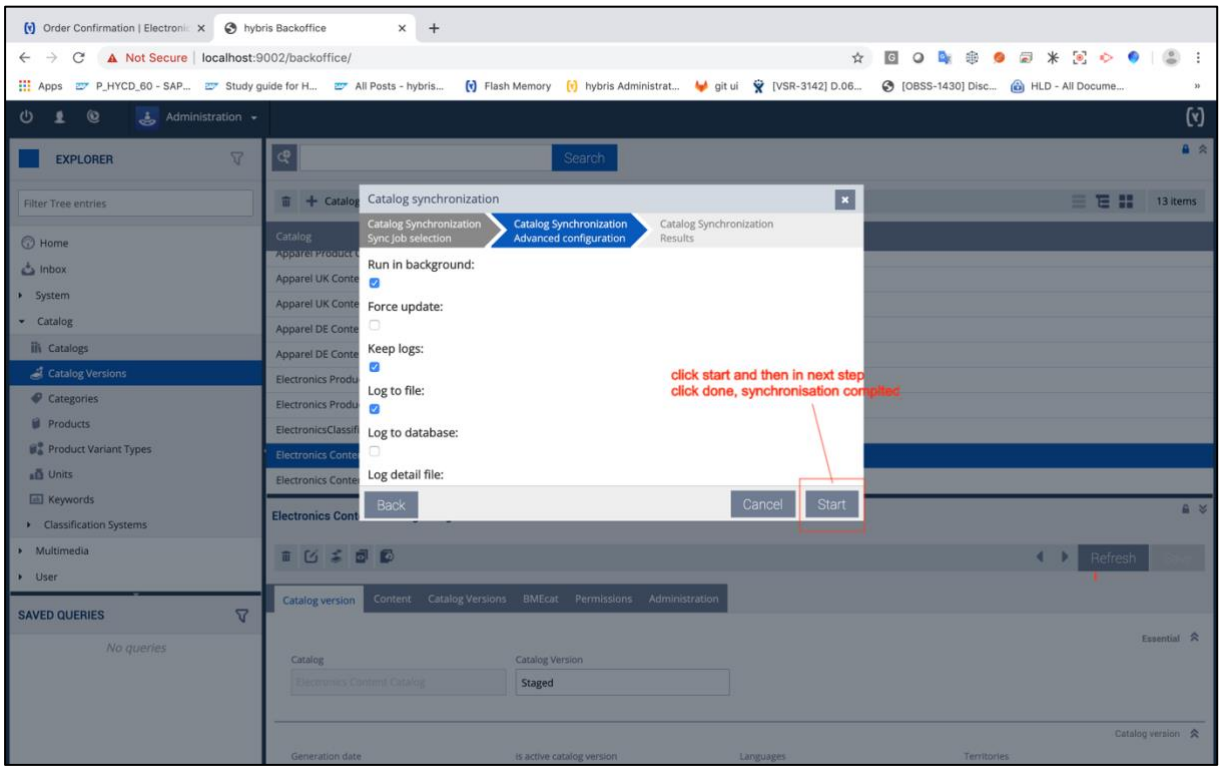


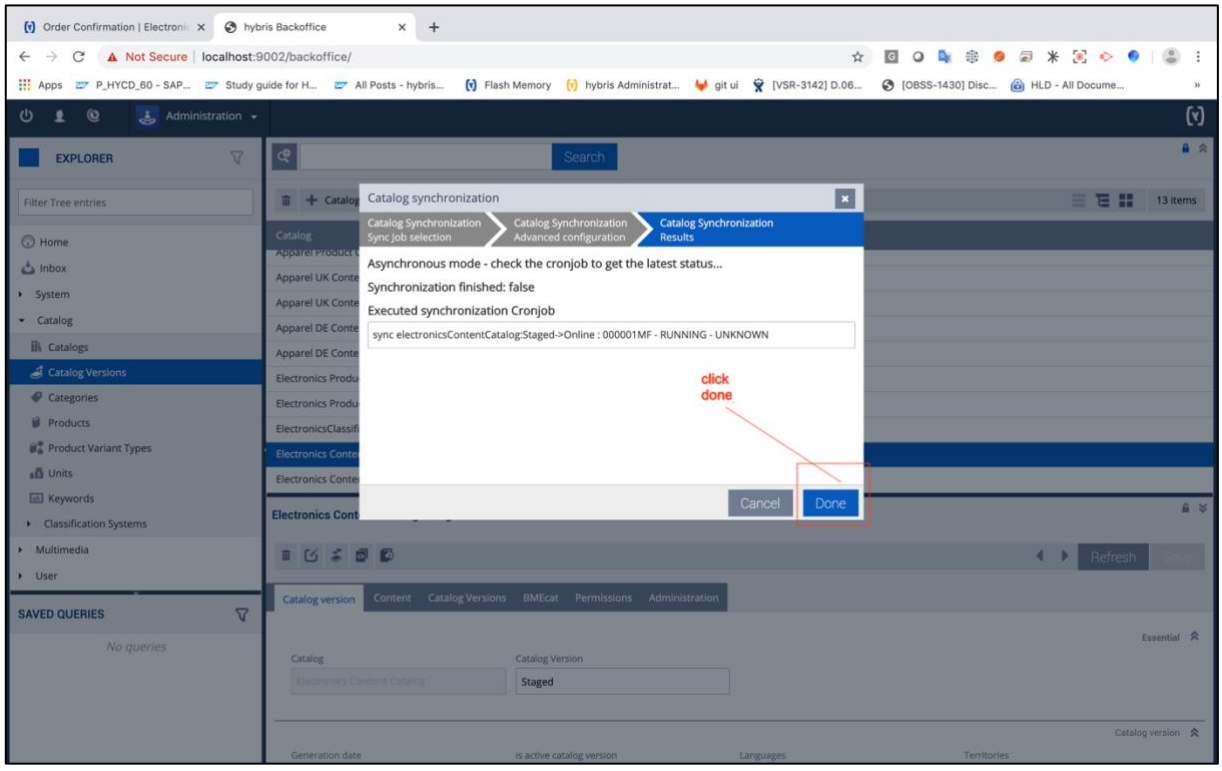
3. **Double click on selected row above and go to Administration tab and change the media**



4. **Similar way the text component can be changed. Now go to catalogs and synchronise the catalog to make changes persist to online version.**




Order Confirmation | Electroni
hybris Backoffice
Not Secure | localhost:9002/backoffice/
Administration
EXPLORER
Search
Catalog synchronization
Catalog Synchronization Sync job selection
Catalog Synchronization Advanced configuration
Catalog Synchronization Results
Run in background:
Force update:
Keep logs:
Log to file:
Log to database:
Log detail file:
click start and then in next step
click done, synchronisation completed
Back
Cancel
Start
Refresh
Catalog version
Content
Catalog Versions
BMEcat
Permissions
Administration
Catalog
Catalog Version
Staged
SAVED QUERIES
No queries


Order Confirmation | Electroni
hybris Backoffice
Not Secure | localhost:9002/backoffice/
Administration
EXPLORER
Search
Catalog synchronization
Catalog Synchronization Sync job selection
Catalog Synchronization Advanced configuration
Catalog Synchronization Results
Asynchronous mode - check the cronjob to get the latest status...
Synchronization finished: false
Executed synchronization Cronjob
sync electronicsContentCatalog:Staged->Online : 000001MF - RUNNING - UNKNOWN
click
done
Cancel
Done
Refresh
Catalog version
Content
Catalog Versions
BMEcat
Permissions
Administration
Catalog
Catalog Version
Staged
SAVED QUERIES
No queries

4. Known Issues

None

5. Versions

Version	Date	Changes
1.0.0	15-May-2020	• Initial release