

Version: 4.7.1

Published date: July 2025

Copyright © 2025 by ESW. All rights reserved. Information contained herein is subject to change without notice.

Embedded Checkout (EC) INTEGRATION GUIDE



**SFCC - ESW Cartridge**

+

Table of Contents

[**1.** **Embedded Checkout** 3](#_Toc189599864)

[2. Order flow 3](#_Toc189599865)

[3. Enable Embedded Checkout on retailer’s website 4](#_Toc189599866)

[3.1. Enable iFrame on ESP via LTI 5](#_Toc189599867)

[3.2. Import metadata and services 8](#_Toc189599868)

[3.3. BM Configurations 10](#_Toc189599869)

[4. Display on storefront 10](#_Toc189599870)

[5. Code Customizations 12](#_Toc189599871)

# **Embedded Checkout**

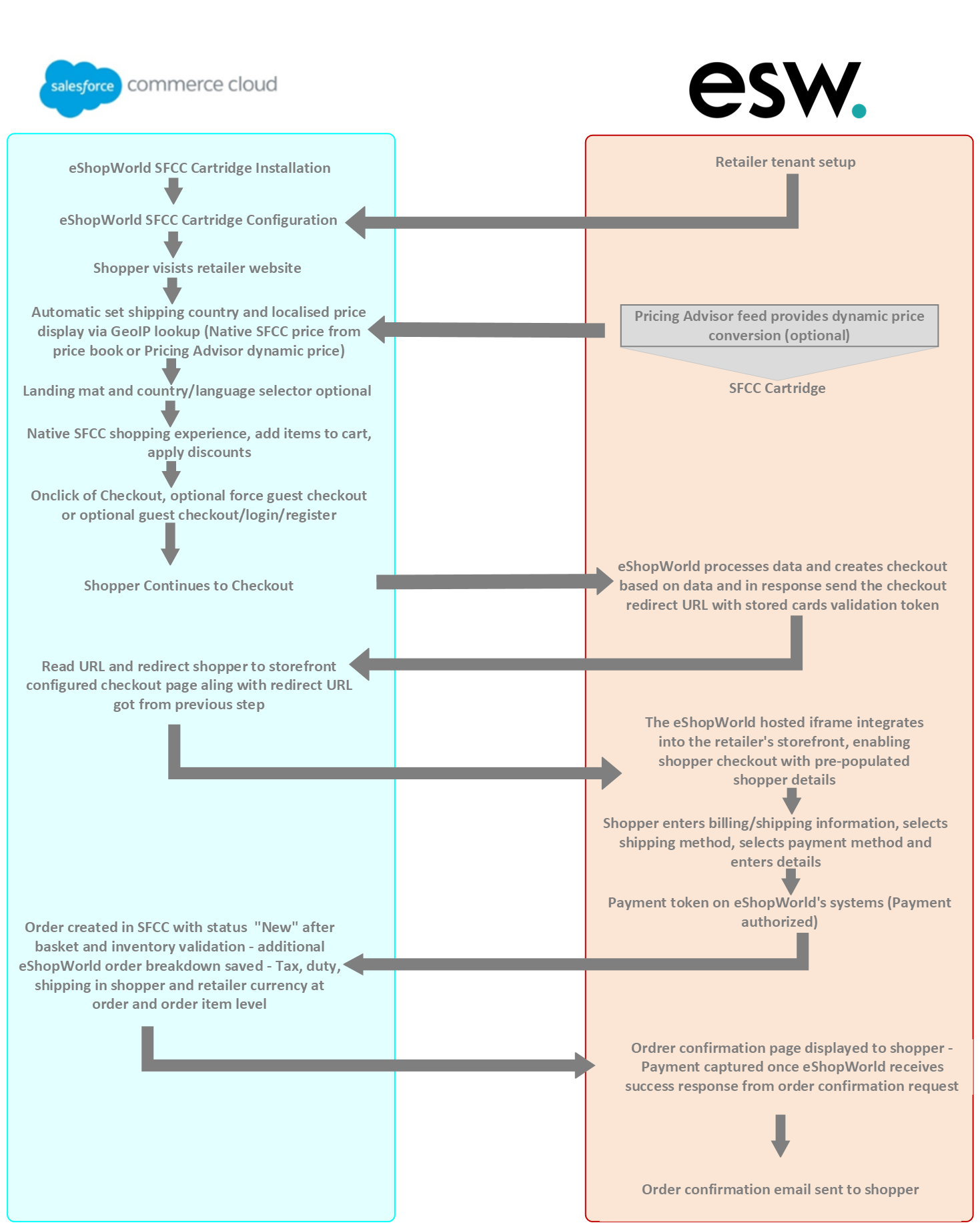
Embedded Checkout allows shoppers to complete their purchases directly within the retailer’s storefront using an iFrame provided by ESW. This feature enhances the shopping experience by keeping the entire checkout process on the retailer’s site, thereby increasing shopper trust and confidence.

Following are the impacted architectures:

* SFRA
* SG
* PWA
* Headless

## Order flow

When embedded checkout is enabled then order will be placed in SFCC only if it is confirmed from ESW (by calling order confirmation webhook from ESW to SFCC). However in order flow without EC the order is placed in SFCC when shopper click on Checkout button on the storefront.

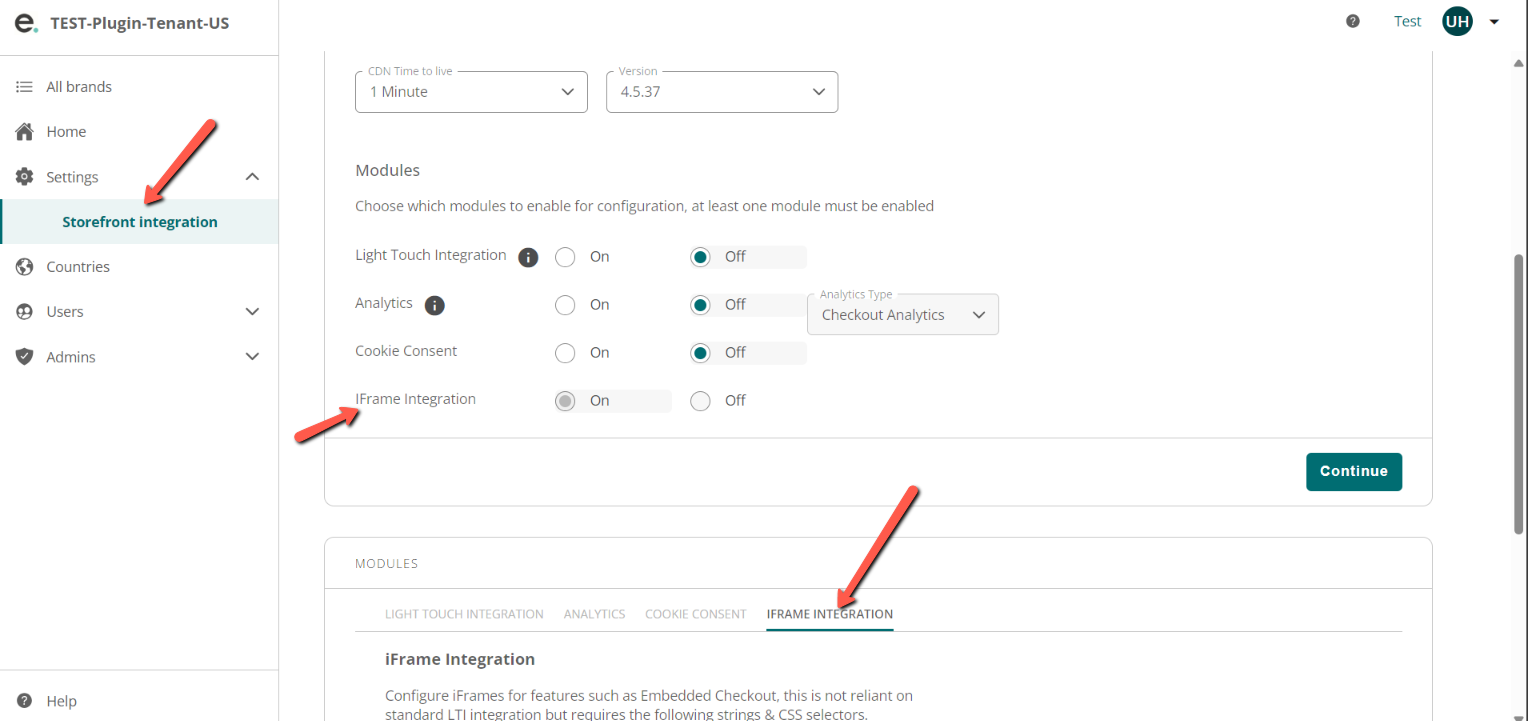
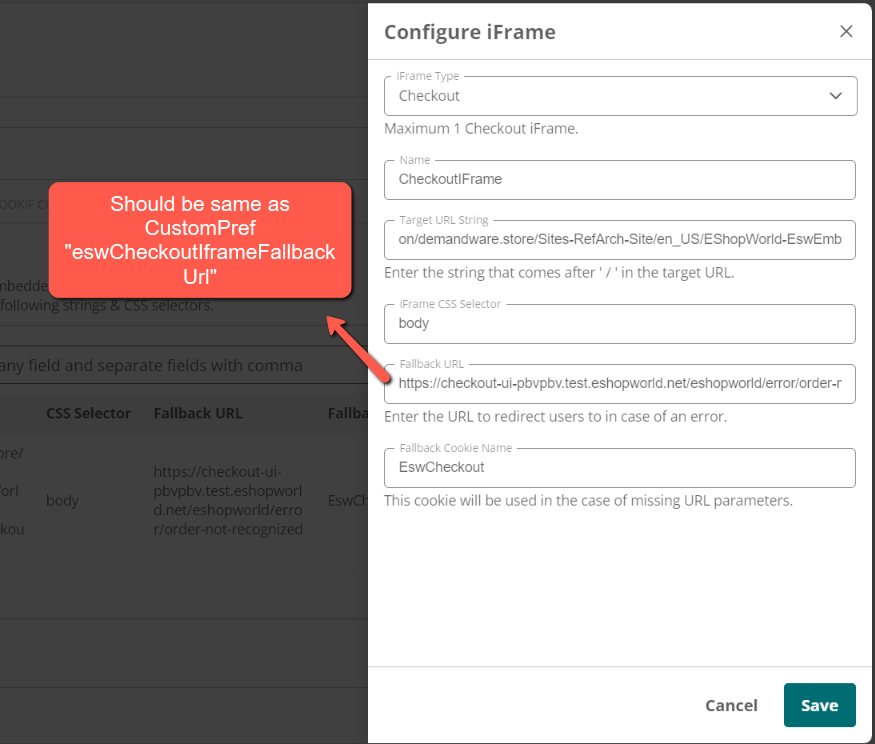


## Enable Embedded Checkout on retailer’s website

To enable Embedded Checkout feature on the storefront, there are few configurations retailer need to add in the SFCC BM. The details of updating configurations is below:

## Enable iFrame on ESP via LTI

It is require to enable iFrame using LTI to get the js which will be responsible to load iFrame and handle all actions performed in iFrame, for example Going back to cart, placing order etc. To add this script please follow these steps:

* Go to ESP panel (link is different for each retailer)
* Enable iframe from Settings -> Storefront Integration -> IFrame Integration  
  
* Create a new slot and select IFrame integration as shown on above screenshot.
* Under this integration there are some settings which are mentioned for test instance in the screenshot. Note that the selector can be any CSS selector where the iframe should be loaded.  
  
* After completing these configurations we will get a link to js script, simply copy that link and paste load in the header section of the page.  
  A screenshot of a computer

  Description automatically generated
* The fallback URL will be called if there is any issue with the iFrame loading, retailers can configure any page for the fallback URL.  
  A screenshot of a computer

  Description automatically generated

## Import metadata and services

**Import Metadata:**

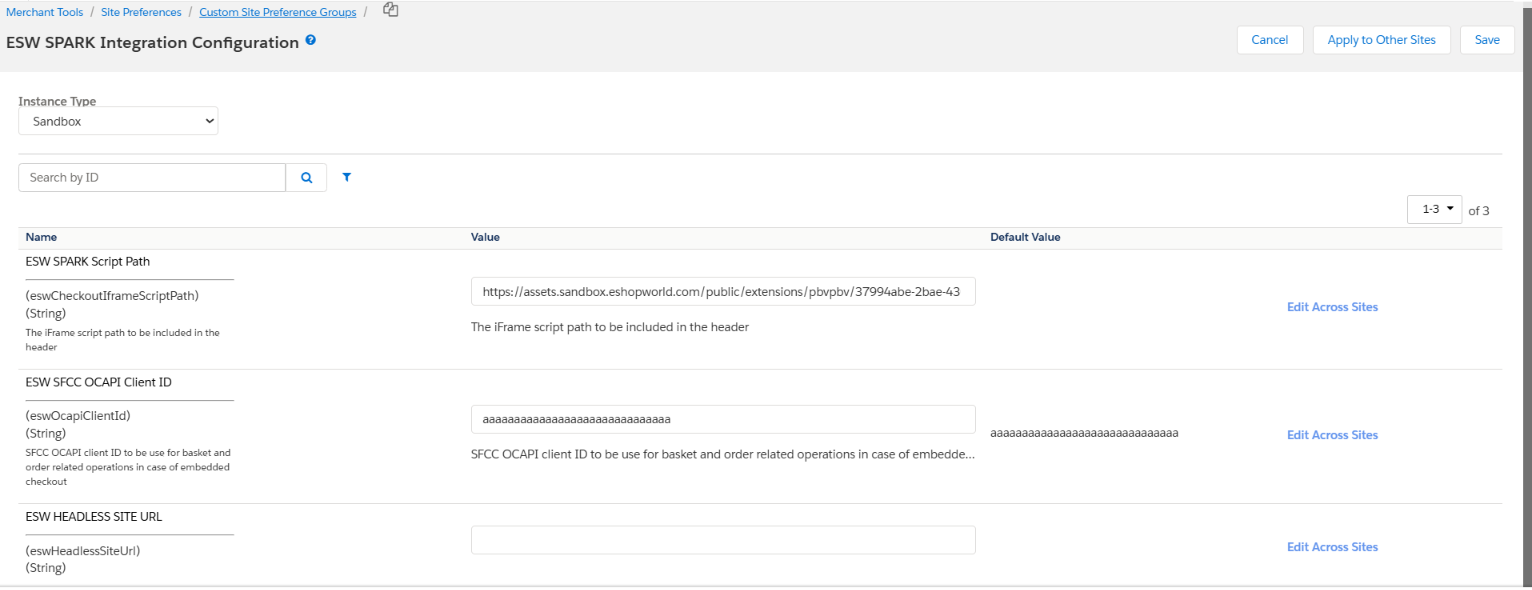
Retailer needs to Import site preference attributes (systemobjects.xml) from **link\_eshopworld\sitesdata\meta.**

After importing meta data, retailer should be able to see following in the SFCC BM

**Custom Preferences:**

“ESW SPARK Integration Configuration” group in Site Preferences along with following attributes:

* **ESW SPARK Script Path:** This is the path of the LTI script, this preference will also act as feature toggle so that if valid path is added then EC will work otherwise shopper will be redirected to ESW
* **ESW SPARK Cookie Name:** The cookie will be used to attempt to retrieve the URL from the cookie if it is not set in the eswiframeurl parameter (For example, in case of browser refresh).
* **ESW SFCC OCAPI Client ID:** This is OCAPI/SCAPI client id of the instance.
* **ESW HEADLESS SITE URL**: This is Site URL of retailer’s Headless domain. Which will be concatenated with LTI script path to redirect the shopper to embedded checkout



**Order level attribute:**

* **ESW Basket UUID:** This new attribute will be added to the Order however it’s not assigned to any group so will not be visible on Order screen.
  + **Attribute ID:** eswBasketUuid
  + Type**: String**
  + **Purpose:** In a preorder, the retailerCartId will be set to basketUUID\_{timeStamp} to prevent any duplication in ESW. This same attribute will be used to cancel the order through an order cancellation call from ESW.

**Import Services:**

Services can be import from (services.xml) from **link\_eshopworld\Metadata**

**After importing services, following services will be displayed:**

* **ESWOrderCreation :** The service is used to obtain the authorization OCAPI token using the Demandware session ID. The URL for the service should be [https://realm.dx.commercecloud.salesforce.com/s/{siteId}/dw/shop/v{ocapi\_version}/](https://realm.dx.commercecloud.salesforce.com/s/%7BsiteId%7D/dw/shop/v%7Bocapi_version%7D/)
* **EswOcapiBasketService:** Service to use basket related oprtations, please setup OCAPI/SCAPI URL of your instance, for example: [https://realm.dx.commercecloud.salesforce.com//s/RefArch/dw/shop/v{OCAPI\_VERSION}/baskets/](https://realm.dx.commercecloud.salesforce.com//s/RefArch/dw/shop/v%7bOCAPI_VERSION%7d/baskets/)
* **EswOcapiOrderService:** Service URL to create order, please enter OCAPI/SCAPI URL of your instance: [https://realm.dx.commercecloud.salesforce.com//s/RefArch/dw/shop/v{OCAPI\_VERSION}/orders](https://realm.dx.commercecloud.salesforce.com//s/RefArch/dw/shop/v%7bOCAPI_VERSION%7d/orders)

## BM Configurations

Following configs needs to be done in the BM to enable the feature:

* Goto “Merchant Tools > Site Preferences > Custom Site Preference Groups > ESW SPARK Integration Configuration” and add following values:
  + **ESW SPARK Script Path:** This is the path of the LTI script.
  + **ESW SPARK Fallback URL:** This is the fallback URL so that in case of iframe failure, shopper will be redirected to this path. Should be same as LTI configurations
  + **ESW SFCC OCAPI Client ID:** This is OCAPI client id of the instance.
  + **ESW HEADLESS SITE URL**: This is Site URL of retailer’s Headless domain. Which will be concatenated with LTI script path to redirect the shopper to embedded checkout
* Goto “Merchant Tools > Site Preferences > Custom Site Preference Groups > ESW Checkout Configuration”
  + Update “ESW Metadata Items” by adding following value:

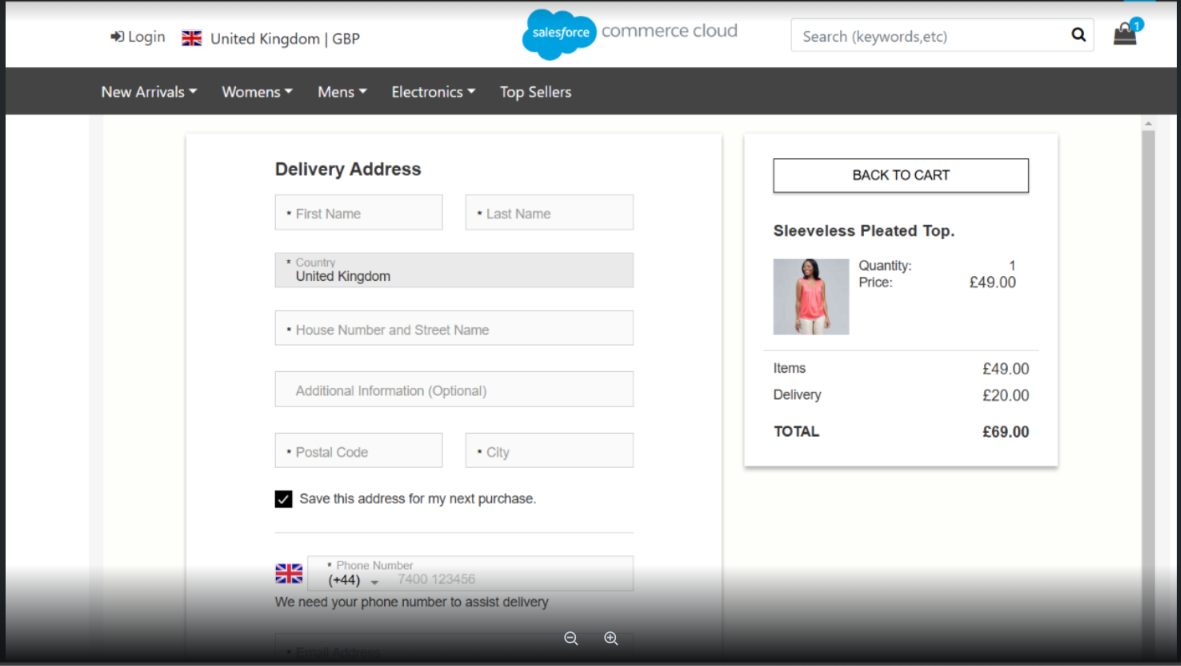
|  |  |
| --- | --- |
| * + - OrderConfirmationUri\_TestOnly|EShopWorld-EswEmbeddedCheckoutNotify |  |

* + - OrderConfirmationBase64EncodedAuth\_TestOnly|1 parameters
* Set cartridge path to:
  + **PWA:** “int\_eshopworld\_pwa:int\_eshopworld\_eckout:int\_eshopworld\_core:app\_storefront\_base:modules”
  + **SFRA:** “int\_eshopworld\_sfra:int\_eshopworld\_eckout:int\_eshopworld\_core:app\_storefront\_base:modules”
  + **SG:** “int\_eshopworld\_controllers:int\_eshopworld\_eckout:int\_eshopworld\_core:app\_storefront\_base:modules”
  + **Headless:** “int\_eshopworld\_headless:int\_eshopworld\_eckout:int\_eshopworld\_core:app\_storefront\_base:modules”
* **Make sure that “int\_eshopworld\_eckout” is available**
* Now BM is ready to use the Embedded Checkout feature.

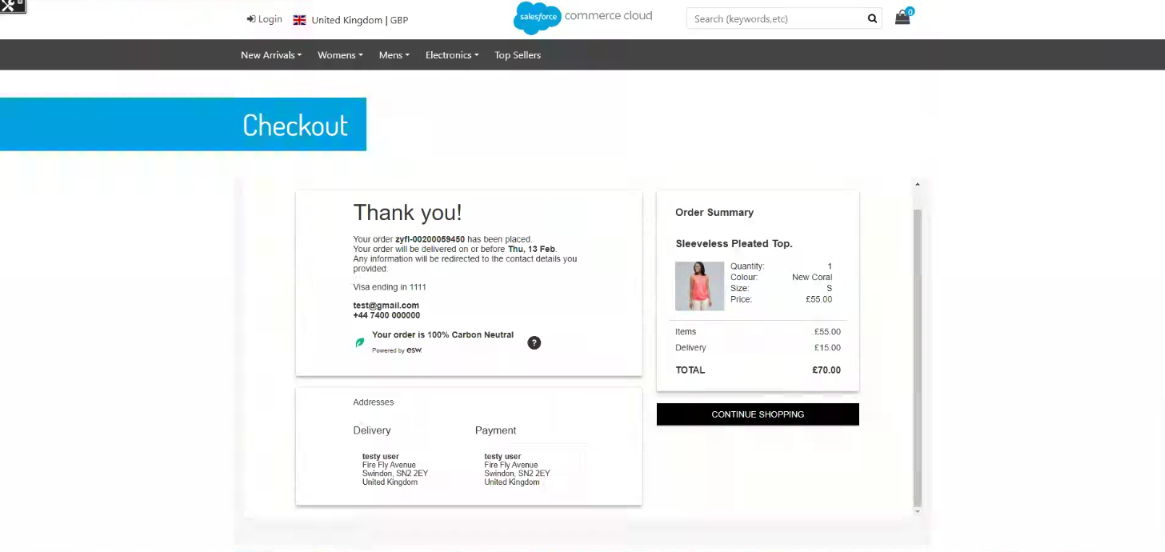
## Display on storefront

On storefront, if EC is enabled, then upon click on the Checkout button the shopper will be redirected to the URL. For each architects following are the URLs

* **SFRA:** [**https://storefront.url/EShopWorld-EswEmbeddedCheckout?eswiframeurl=checkoutUrlFromPreOrder**](https://storefront.url/EShopWorld-EswEmbeddedCheckout?eswiframeurl=checkoutUrlFromPreOrder)
* **SG:** [**https://storefront.url/EShopWorldSG-EswEmbeddedCheckout?eswiframeurl=checkoutUrlFromPreOrder**](https://storefront.url/EShopWorldSG-EswEmbeddedCheckout?eswiframeurl=checkoutUrlFromPreOrder)
* **PWA:** [**https://storefront.url/esw-checkout?eswiframeurl=checkoutUrlFromPreOrder#**](https://storefront.url/esw-checkout?eswiframeurl=checkoutUrlFromPreOrder)
* **HeadlessSFRA:** [**https://storefront.url/EShopWorld-EswEmbeddedCheckout?eswiframeurl=checkoutUrlFromPreOrder**](https://storefront.url/EShopWorld-EswEmbeddedCheckout?eswiframeurl=checkoutUrlFromPreOrder)
* **HeadlessSG:** [**https://storefront.url/EShopWorldSG-EswEmbeddedCheckout?eswiframeurl=checkoutUrlFromPreOrder**](https://storefront.url/EShopWorldSG-EswEmbeddedCheckout?eswiframeurl=checkoutUrlFromPreOrder)

Please note that the query param “eswiframeurl” is static and can not be change. The preorder Url will removed from the URL through LTI script. The final display of the iFrame is shown: 

After completing the the payment the order confirmation will looks like:



**Note:** If, for any reason, the iframe fails to load on the shopper’s browser, the page will be redirected to the native ESW checkout as a fallback.

## Code Customizations

To implement embedded checkout, a few customizations are required in the code base**. If you need to copy the code, please do so from the provided cartridge instead of the Word file**. The details are as follows:

**SFRA:**

* No customization is required for SFRA.

**PWA:**

* Open “link\_eshopworld\esw-pwa-app\overrides\app\routes.jsx”
  + Import the “EswEmbeddedCheckout” component

import EswEmbeddedCheckout from './pages/esw-embedded-checkout'

* + Add route with following configuration

    {

        path: '/esw-checkout',

        component: EswEmbeddedCheckout,

        exact: true

    }

* For detailed PWA integration please follow the docs “eShopWorld\_Cartridge\_Pwa\_Integration.docx”

**SG:**

* Open “/link\_eshopworld/cartridges/int\_eshopworld\_controllers/cartridge/controllers/EShopWorldSG.js”
  + Require following 
  + Update “function preOrderRequest()”A screenshot of a computer

    Description automatically generated
  + Add following:

/\*\*

 \* Function to handle Embeded Checkout order confirmation request in V2

 \*/

function eswEmbeddedCheckoutNotify() {

    let eswCoreApiHelper = require('\*/cartridge/scripts/helper/eswCoreApiHelper'),

        eswCoreService = require('\*/cartridge/scripts/services/EswCoreService').getEswServices(),

        eswOcapiServiceHelper = require('\*/cartridge/scripts/services/EswOcapiService').getEswOcapiServices,

        embCheckoutHelper = require('\*/cartridge/scripts/helper/eckoutHelper').eswEmbCheckoutHelper;

    let Transaction = require('dw/system/Transaction'),

        logger = require('dw/system/Logger'),

        ProductMgr = require('dw/catalog/ProductMgr'),

        responseJSON = {};

    let preorderServiceObj = eswCoreService.getSFCCOcapi();

    let currentBasketData = null;

    if (eswHelper.getBasicAuthEnabled() && !request.httpHeaders.authorization.equals('Basic ' + eswHelper.encodeBasicAuth())) {

        response.setStatus(401);

        logger.error('ESW Order Confirmation Error: Basic Authentication Token did not match');

    } else {

        try {

            let obj = JSON.parse(request.httpParameterMap.requestBodyAsString);

            eswHelper.eswInfoLogger('Esw Order Confirmation Request', JSON.stringify(obj));

            let ocHelper = require('\*/cartridge/scripts/helper/orderConfirmationHelper').getEswOcHelper();

            let inventoryValidationPlis = { productLineItems: new ArrayList([]) };

            let dwSid = eswHelper.getDwsid(obj);

            let basketID = eswHelper.getRetailerCartId(obj.retailerCartId);

            let customerEndpoint = Constants.CUSTOMER\_AUTH + eswHelper.getOcapiClientID();

            let customerAuth = preorderServiceObj.call({ requestBody: { type: 'session' }, endpoint: customerEndpoint, dwsid: dwSid });

            let authKey = customerAuth.ok && !empty(customerAuth.object) ? customerAuth.object.getResponseHeader('authorization') : null;

            let shopperCurrency = ('checkoutTotal' in obj) ? obj.checkoutTotal.shopper.currency : obj.shopperCurrencyPaymentAmount.substring(0, 3);

            ocHelper.setOverridePriceBooks(obj.deliveryCountryIso, shopperCurrency);

            let ocapiBasketResponse = eswOcapiServiceHelper.ocapiBasketService().call({

                basketId: basketID,

                httpMethod: 'GET',

                authToken: authKey

            });

            if (ocapiBasketResponse.isOk() && !empty(ocapiBasketResponse.getObject())) {

                currentBasketData = JSON.parse(ocapiBasketResponse.object.text);

                if (eswHelper.getEnableInventoryCheck()) {

                    for (let i = 0; i < currentBasketData.product\_items.length; i++) {

                        inventoryValidationPlis.productLineItems.add({

                            product: ProductMgr.getProduct(currentBasketData.product\_items[i].product\_id),

                            quantityValue: currentBasketData.product\_items[i].quantity

                        });

                    }

                    if (!ocHelper.validateEswOrderInventory(inventoryValidationPlis)) {

                        throw new Error('Inventory validation failed');

                    }

                }

            }

            let isValidBasket = eswCoreApiHelper.compareBasketAndOcProducts(currentBasketData, obj);

            if (!isValidBasket) {

                throw new Error('Basket data from OCAPI and ESW Checkout are not equal');

            }

            let ocapiOrderResponse = eswOcapiServiceHelper.ocapiOrderService().call({

                httpMethod: 'POST',

                authToken: authKey,

                basketId: basketID,

                countryCode: obj.deliveryCountryIso

            });

            if (ocapiOrderResponse.isOk() && !empty(ocapiOrderResponse.getObject())) {

                let orderData = JSON.parse(ocapiOrderResponse.object.text);

                Transaction.wrap(function () {

                    embCheckoutHelper.processUpdateOrderAttributes(orderData.order\_no, obj, null);

                });

                responseJSON = {

                    OrderNumber: orderData.order\_no,

                    EShopWorldOrderNumber: obj.eShopWorldOrderNumber,

                    ResponseCode: '200',

                    ResponseText: 'Success'

                };

            } else {

                throw new Error('Error while generating order from the basket');

            }

        } catch (e) {

            logger.error('ESW Service Error: {0}', e.message);

            if (e.name === 'SystemError') {

                response.setStatus(429);

                responseJSON.ResponseCode = '429';

                responseJSON.ResponseText = 'Transient Error: Too many requests';

            } else {

                response.setStatus(400);

                responseJSON.ResponseCode = '400';

                responseJSON.ResponseText = 'Error: Internal error';

            }

        }

        eswHelper.eswInfoLogger('Esw Order Confirmation Response', JSON.stringify(responseJSON));

    }

    Response.renderJSON(responseJSON);

}

* + Add following:

/\*\*

 \* Function to be called to render ESW Embeded Checkout page

 \*/

function eswEmbeddedCheckout() {

    app.getView({

        eswCheckoutUrl: request.httpQueryString.eswiframeurl

    }).render('checkout/eswEmbeddedCheckout');

}

* + Add routes:

/\*\* Renders the Embeded checkout

 \* @see module:controllers/EShopWorldSG~eswEmbeddedCheckout \*/

exports.EswEmbeddedCheckout = guard.ensure(['get'], eswEmbeddedCheckout);

/\*\* Handles the order confirmation request

 \* @see module:controllers/EShopWorldSG~EsWEmbeddedCheckoutNotify \*/

exports.EsWEmbeddedCheckoutNotify = guard.ensure(['post'], eswEmbeddedCheckoutNotify);

* Add template “/link\_eshopworld/cartridges/int\_eshopworld\_controllers/cartridge/templates/default/checkout/eswEmbeddedCheckout.isml”