CardSafe Payment Widget

Installation and Usage

2022

Contents

History

Date	Author	Description
2021-02-15	Darren Jerrard	First Draft
2021-08-26	Brett Mair	Revised with CSS data attributes on pay button
2021-11-18	Darren Jerrard	Updated with ConligoPayWidgetConnector.dll
2022-02-08	Heather Keep	Updated path to payment widget script and css files.
2022-02-16	Darren Jerrard	Updated with SessionID persistence and AvsCvvResults information.
2022-05-31	Heather Keep	Updated with samples, reorganized structure, OWTRX information
2022-06-15	Heather Keep	Updated with Bootstrap 5x support details, exclude credit card data configuration feature
2022-06-30	H. Keep	Updated with customization of card validation, removed instructions on how to write to the OWTRX table – this is now a built-in feature of Conligo Pay
2022-09-15	Darren Jerrard	Updated section on retrieving a card list to be displayed in the dropdown.

Conligo (formerly Iciniti Corporation)

353 Dufferin Avenue
London, Ontario,
Canada N6B 1Z5
Tel: (519) 432-9599
Support: support@conligo.ca
Web site: http://www.conligo.ca

Introduction

The purpose of this document is to describe how to integrate an e-commerce web site to the Card Safe Payment Widget.

This document is intended for use by web application developers.

Overview

There are two main components to the integration, the 'Pay' button and the payment handler. The Pay button is an HTML element while the payment handler is typically some server-side code in whatever language drives your web site.

Clicking the 'Pay' button will prompt for credit card information. When the card information has been entered, a JavaScript handler you create and specify on the Pay button will receive a response indicating if it's ok to continue processing.

Without the Pay button, your existing workflow for the order submission probably marks the order as ready for processing such as sending an email confirmation, shipping, etc. Whatever the behaviour, adding the Pay button means you will need to update that handling to include performing the payment transaction.

This is done in a couple of steps. The first step is to have a 'responseHandler' function that receives the response from the Pay button being clicked. This response comes after some card validation is run and will tell you if it's ok to proceed with the transaction request. If it was successful, then you must check if a 3D-Secure challenge is required (Moneris only). This is done by checking the boolean value in the MustChallenge field of the response object.

If no 3D-Secure challenge is required, then the transaction has been executed and you can proceed to check if the transaction was successful before continuing with the regular processing of the order. If the transaction failed, you could direct the browser to an error page and have the customer try again, perhaps with a different card.

If a 3D-Secure challenge is required, the response object contains an HTML form that you must present to the browser. This form contains a return URL that the browser will redirect to when the challenge has completed, whether successful or failed. You specify that return URL when you define the attributes in the Pay button's HTML.

Card Safe URL

There are two URLs for Conligo Pay's Card Safe web service. Make sure you use the correct URL. The URL must be set in the 'Pay' button (described below) in the 'data-conligo-cardsafeurl' attribute.

Production URL: https://cardsafe.conligo.ca

Test URL: https://testcardsafe.conligo.ca

The 'Pay' Button

The Pay button is an HTML element. It can be an actual <button> element, a , or another type of element. It **must** be marked with a CSS class name of "CardSafePayButton".

e.g. <button type="button" class="CardSafePayButton">Pay Now</button>

To enable this button, you must link to its CSS and JavaScript library. Choose your Bootstrap version accordingly.

Bootstrap version 4

```
<link href="https://cardsafe.conligo.ca/css/payment_widget/payment_widget.css" rel="stylesheet"/>
<script src="https://cardsafe.conligo.ca/javascript/payment_widget/payment_widget.js"
        type="text/javascript"
        defer="defer" ></script>
```

Bootstrap version 5

```
<link href="https://cardsafe.conligo.ca/css/payment_widget/payment_widget.bs5.css" rel="stylesheet"/>
<script src="https://cardsafe.conligo.ca/javascript/payment_widget/payment_widget.bs5.js"
        type="text/javascript"
        defer="defer" ></script>
```

If you're running on the test system, use 'testcardsafe.conligo.ca' instead of 'cardsafe.conligo.ca' in the links above.

This library depends on jQuery, so be sure to load jQuery first before the <script> tag.

Clicking the Pay button will display the Card Entry window. If this window doesn't appear, check that you are using the correct URL and that the 'defer' attribute is specified on the <script> tag above.

This window captures the card information but does not execute the transaction. Instead, a preliminary check is done to see if the card is enrolled in 3D-Secure (Moneris only) and a response is returned to your JavaScript handler.

e.g., <button type="button" class="CardSafePayButton"

```
                data-conligo-cardsafeurl="<url>"
                data-conligo-responsehandler="your_response_handler_name"
```

>Pay Now</button>

When you specify your handler's name, do not include the parentheses.

I.e., "myhandler" and not "myhandler()"

Your response handler must accept a single parameter. This parameter is the response from the Card Entry form's submission. If it returns a Result of "Success", then you may continue to payment submission.

In the following sections we will describe how to process a transaction and configure the pay button.

Handling the PayButton Response

Your response handler will look something like this:

```
function responseHandler(response) {

    if (response.Result === "Success") {

        if (response.MustChallenge) {
            $("#3dsFormContainer").append(response.ChallengeForm);
             $("#3dsFormContainer").find("form").submit();                }
        else {
            window.location.href = <page that will process the trx response>
        }
    }
    else {
            console.log(response.ErrorMessage);
            // Display the error, ask for another card, etc.
        }
}
```

If Result is not “Success”, then there was an error, and the error message can be found in the ErrorMessage property.

If the Result is “Success”, you can continue with your form’s submission as you normally would have without the Pay button. Presumably, your form finalizes order information in some fashion. That part of the workflow will need to be modified to do the payment transaction.

The window.location.href value is the relative URL to the page that will process the transaction. It needs to accept the TrxId value from the response object.

For example, a Conligo eCommSuite web store would set window.location.href to:

```
"Payment/WebPayments/TransactionComplete.aspx?trxid=" + response.TrxId
```

Performing the Transaction

ConligoPay Widget Connector

To simplify some of the coding, you'll use our ConligoPayWidgetConnector library. It is a small .NET Standard 2.0 library that can be used in .NET Framework, .NET Core, .NET 5 & 6 projects.

To instantiate it:

```
using com.conligo.widget.connector;
var connector = new WidgetConnector(urlForCardSafe, tenantId, Request.Headers, Request.Params);
```

Where:

- urlForCardSafe = The Card Safe URL (either the test or production URL for Card Safe)
- tenantId = your assigned Tenant ID.
- Request = The Headers and Params values from the System.Web.HttpRequest.

These values must be passed to the WidgetConnector's constructor.

Widget Connector Example

```
using com.conligo.widget.connector;
…

PaymentStatusInfo paymnentStatusInfo = null;

var connector = new WidgetConnector(CardSafeUrl, tenantId, Request.Headers, Request.Params);

  if (connector.OkToProceed)
  {
       paymnentStatusInfo = connector.TryProcessTransaction();

       if (psi.Approved)
       {
           // store these values and continue with order processing
           string trxReference = psi.GatewayReference;
           string approvalCode = psi.ApprovalCode;
       }
       else
       {
             // Redirect to a failure page or back to attempting another payment
             string error = psi.ErrorDescription;
       }
  else
  {
        // cardholder was challenged, but failed
    // consider asking for another card
  }
```

Processing the Response

The TrxId value returned in the response (`response.TrxId`) identifies the transaction within the ConligoPay Card Safe system. Likewise, the GatewayReference value provided by the PaymentStatusInfo object (`PaymentStatusInfo.GatewayReference`) identifies the transaction within the payment gateway's system. These values are unique to each system and not interchangeable.

It is not strictly necessary for you to store both values, but we recommend that you store at least one of them. If you're going to use the Conligo Order Reader to pull the order data into your ERP system, then you must have the TrxId value. The Order Reader uses this value to retrieve the transaction details from Card Safe when writing payment information into the ERP system for use by the Conligo Pay Sage 300 modules.

Similarly, if you are going to use your own custom integration but wish to use the Conligo Pay Sage 300 module for subsequent processing, the TrxId value is necessary for the same reason: to retrieve the transaction details from Card Safe when writing payment information into the ERP system for use by the Conligo Pay Sage 300 modules.

Exclude Card Details from the Response

If you wish to exclude details related to the credit card information related to the transaction, be sure to set the property on the connector object before you process the transaction. This will ensure that Card Safe clears out the cardholder, card type, masked card, and expiration date from the response before it returns:

```
var connector = new WidgetConnector(CardSafeUrl, tenantId, Request.Headers, Request.Params);
connector.ExcludeCardInformationFromCustomerTransactionRecord = true;
```

Configuring the Pay Button

Identification

The Pay button needs to know about your merchant account which must already have been configured by Conligo. For this, you must specify your Tenant ID and Account Selector. You must also provide an ID for the customer. This customer ID must not be shared between multiple customers of your ERP system.

```
<button id="btnPayWithWidget" type="button"
      data-conligo-cardsafeurl="<test or production url>"
      data-conligo-tenantid="<tenant id assigned by Conligo>"
      data-conligo-accountselector="<indicates the gateway account to use>"
      data-conligo-customerid="<a value to uniquely identify the cardholder>"
      data-conligo-responsehandler="responseHandler" class="CardSafePayButton">Pay Now
</button>
```

Payment Information

The Pay button needs to know the URL to redirect to in order to process the payment result, the total amount of the order, the currency of the transaction, the type (Auth or Sale) and finally a merchant reference. This could be the order or quote number.

```
data-conligo-trxcompleteurl="<url that handles the payment result>"
data-conligo-amount="<dollar amount, using '.' for cents separator>"
data-conligo-currencycode="<currency of the transaction>"
data-conligo-transactiontype="Auth"  (or "Sale")
data-conligo-merchantreference="<order or invoice number>"
```

See Appendix B “Session Persistence when Challenged by 3D-Secure” section below about adding “sid=<%=Session.SessionID%>” to the data-conligo-trxcompleteurl attribute.

Address Information

With the **Bootstrap 5.x** version of the widget, it’s possible to pre-populate the address fields and to hide those fields if required. This is achieved by setting the following toggle:

```
data-conligo-show-address-fields="<true or false>"
```

and subsequent address field value attributes on the widget:

```
data-conligo-address-line-1-value="<your customer street address>"
data-conligo-address-city-value="<your customer city>"
data-conligo-address-region-value="<your customer region/state/province>”
data-conligo-address-country-value="<your customer two-character ISO country code>"
data-conligo-address-postal-code-value="<your customer postal/zip code>"
```

Bootstrap 4.x Attributes

Labels

You can change the labels that appear by setting the *data-conligo-formtitle* and *data-conligo-xxxlabel* values (see highlighted region):

```
data-conligo-formtitle="Card Information"
data-conligo-cardnumberlabel="Card Number"
data-conligo-cardholderlabel="Cardholder"
data-conligo-expirationlabel="Exp. Date"
data-conligo-securitycodelabel="CVV"
data-conligo-submitbuttonlabel="Proceed
```

CSS Classes / Customizing the Layout

To customize the default plain HTML layout, enter your desired CSS class names in the appropriate data-conligo-xxx attribute from the previous section:

```
data-conligo-header-row="<Your Custom CSS Classes>"
data-conligo-row="<Your Custom CSS Classes>"
data-conligo-button="<Your Custom CSS Classes>"
data-conligo-label="<Your Custom CSS Classes>"
data-conligo-input="<Your Custom CSS Classes>"
data-conligo-select="<Your Custom CSS Classes>"
data-conligo-rule="<Your Custom CSS Classes>"
```

Modal Dialog Attributes

The data-conligo-modal-xxx attributes pertain to the use of Bootstrap 4.x and can be regarded as placeholders for the respective Bootstrap modal divs. Bootstrap 4.x is required. For more information about bootstrap, visit https://getbootstrap.com to learn more about this CSS framework.

```
data-conligo-modal-content="<Your Custom CSS Classes>"
data-conligo-modal-header="<Your Custom CSS Classes>"
data-conligo-modal-body="<Your Custom CSS Classes>"
data-conligo-modal-backdrop="<Your Custom CSS Classes>"
data-conligo-modal-dialog="<Your Custom CSS Classes>"
data-conligo-modal-footer="<Your Custom CSS Classes>"
```

Example Pay Button Bootstrap 4x

```
<button type="button" id="conligoPayButton"
        onclick="setTimeout('finalize()',500);return true;"
        data-conligo-tenantid="@tenantId"
        data-conligo-accountselector="@Model.CurrencyCode"
        data-conligo-customerid="@Model.CustomerNumber"
        data-conligo-cardsafeurl="@cardSafeUrl"
        data-conligo-trxcompleteurl="@Url.Action("TransactionComplete", "WebPayment", routeValues: new { orderNumber=Model.OrderNumber, accountSelector=Model.CurrencyCode, sid=Session.SessionID })"
        data-conligo-amount="@Model.TotalWithTax"
        data-conligo-currencycode="@Model.CurrencyCode"
        data-conligo-transactiontype="@transactionType"
        data-conligo-merchantreference="@tempOrderNumber"
        data-conligo-formtitle=""
        data-conligo-cardnumberlabel="Card Number"
        data-conligo-cardholderlabel="Cardholder Name"
        data-conligo-expirationlabel="Card Expires"
        data-conligo-securitycodelabel="Security Digits"
        data-conligo-submitbuttonlabel="Submit"
        data-conligo-header-row=""
        data-conligo-row="conligo-row"
        data-conligo-button=""
        data-conligo-label="form-text small"
        data-conligo-input="form-control"
        data-conligo-select="form-control custom-select"
        data-conligo-rule="d-none"
        data-conligo-modal-backdrop="my-backdrop"
        data-conligo-modal-dialog="modal-lg"
        data-conligo-modal-header="border-bottom-0"
        data-conligo-modal-content=""
        data-conligo-modal-body=""
        data-conligo-modal-footer="border-top-0"
        data-conligo-responsehandler="responseHandler"
        class="btn btn-secondary my-1 border CardSafePayButton">
    <span>Enter Credit Card Information</span>
</button>
```

Figure 1 Example html button implementation

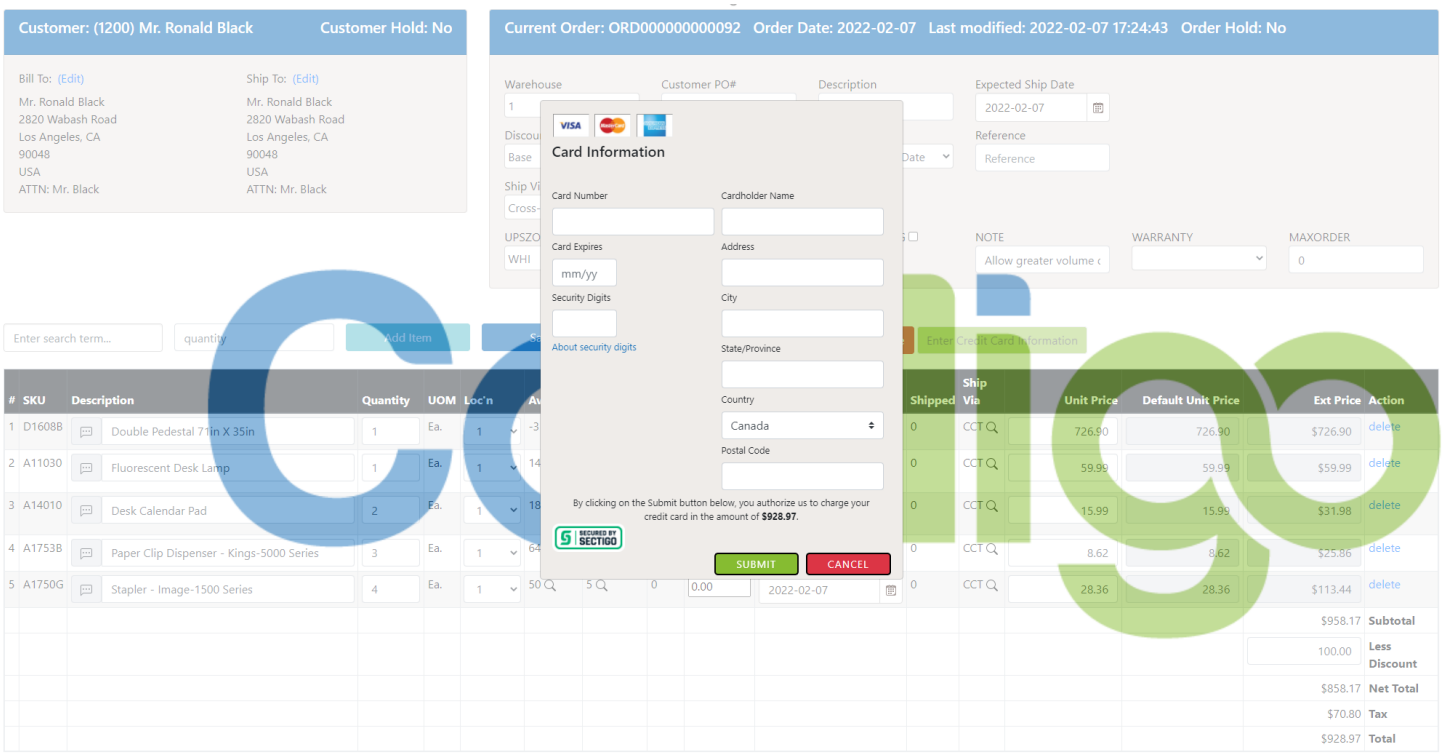

Figure 2 Example Bootstrap 4.x widget implementation with underlay

Bootstrap v5.x Attributes

Card and Button Labels

You can change the labels that appear by setting the *data-conligo-title* and *data-conligo-xxx-label* values:

```
data-conligo-title="Card Information"
data-conligo-card-number-label="Card Number"
data-conligo-cardholder-label="Cardholder"
data-conligo-card-expiration-label="MM/YY"
data-conligo-security-code-label="CVV"
data-conligo-pay-button-label="Pay Now"
data-conligo-close-button-label="Cancel"
```

Address Field Labels

If choosing to display the address fields, you can change the labels that appear by setting the following *data-conligo-xxx-label* values:

```
data-conligo-address-line-1-label="Address"
data-conligo-address-city-label="City"
data-conligo-address-region-label="Region"
data-conligo-address-country-label="Country"
data-conligo-address-postal-code-label="Postal Code"
```

CSS Classes / Customizing the Layout

To customize the default plain HTML layout, enter your desired CSS class names in the appropriate data-conligo-**css**-xxx attribute:

```
data-conligo-css-header-row="<Your Custom CSS Classes>"
data-conligo-css-form="<Your Custom CSS Classes>"
data-conligo-css-row="<Your Custom CSS Classes>"
data-conligo-css-button="<Your Custom CSS Classes>"
data-conligo-css-button-row="<Your Custom CSS Classes>"
data-conligo-css-label="<Your Custom CSS Classes>"
data-conligo-css-input="<Your Custom CSS Classes>"
data-conligo-css-select="<Your Custom CSS Classes>"
data-conligo-css-rule="<Your Custom CSS Classes>"
```

CSS Classes / Customizing Address fields and Card data fields

You can change the individual style of card and address fields by changing the *data-conligo-**css**-xxx values:*

```
data-conligo-css-card-number=""
data-conligo-css-cardholder=""
data-conligo-css-card-expiration=""
data-conligo-css-security-code=""
data-conligo-css-address-line-1=""
data-conligo-css-address-city=""
data-conligo-css-address-country=""
data-conligo-css-address-region=""
data-conligo-css-address-postal-code=""
```

CSS Classes / Customizing Address field Labels and Card data field Labels

You can change the individual style of card and address field labels by changing the *data-conligo-**css**-xxx-**label** values:*

```
data-conligo-css-card-number-label="<Your Custom CSS Classes>"
data-conligo-css-cardholder-label="<Your Custom CSS Classes>"
data-conligo-css-card-expiration-label="<Your Custom CSS Classes>"
data-conligo-css-security-code-label="<Your Custom CSS Classes>"
data-conligo-css-address-line-1-label="<Your Custom CSS Classes>"
data-conligo-css-address-city-label="<Your Custom CSS Classes>"
data-conligo-css-address-country-label="<Your Custom CSS Classes>"
data-conligo-css-address-region-label="<Your Custom CSS Classes>"
data-conligo-css-address-postal-code-label="<Your Custom CSS Classes>"
```

CSS Classes / Customizing Pay and Close Buttons

You can change the individual style of the pay and close buttons by changing the *data-conligo-**css**-xxx-**button** values:*

```
data-conligo-css-pay-button="<Your Custom CSS Classes>"
data-conligo-css-close-button="<Your Custom CSS Classes>"
```

CSS Classes / Customizing the Modal Dialog

You can change the individual style of modal dialog by changing the *data-conligo-**css-modal**-xxx values:*

```
data-conligo-css-modal-header="<Your Custom CSS Classes>"
data-conligo-css-modal-content="<Your Custom CSS Classes>"
data-conligo-css-modal-body="<Your Custom CSS Classes>"
```

Card Validation / Customizing the Tooltips and Error messages

You can change the content of the messages displayed to the users when card data entered does not pass validation:

```
data-conligo-tooltip-cardnumber="<Your custom tooltip>"
data-conligo-errmsg-badcardnumber="<Your custom error message>"
data-conligo-errmsg-unknowncardtype="<Your custom error message>"
```

Card Selection

By default, the widget requires the user to re-enter their card information. You can have the widget display a list of previously vaulted cards. Users will still need to enter the CVV as it cannot be stored. To make use of this feature, the following attributes must be set:

```
data-conligo-show-card-number-list="true"
data-conligo-card-number-list="<see below>"
```

When enabled, a drop-down list of available cards is displayed. To customize the label for this list use:

```
data-conligo-card-number-list-label="Select a Card"
```

The value set into the data-conligo-card-number-list attribute comes from the WidgetConnector class. You can retrieve the list as follows.

In an ASP.NET page's Page_Load() function, for example:

```
WidgetConnector widgetConnector = new WidgetConnector(url, tenantId);
var cardListResult = await widgetConnector.GetCardListAsync(accountSelector, customerId);
Session["CardList"] = cardListResult.Success ? cardListResult.CardList : "";
```

Then you'd set the attribute like this:

```
data-conligo-card-number-list="<%= Session["CardList"] %>"
```

Likewise, in an MVC app, you could return the value from cardListResult.CardList in the ViewBag and reference it in the attribute:

```
ViewBag.CardNumberList = cardListResult.Success ? cardListResult.CardList : "";
```

Then you'd set the attribute like this:

```
data-conligo-card-number-list="@ViewBag.CardNumberList"
```

If cardListResult.Success is false, then you should find an error message in cardListResult.ErrorDescription. The Widget can still be used, it just won't have cards displayed in the dropdown.

If you're not in an async method, you can call widgetConnector.GetCardList(…) instead of widgetConnector.GetCardListAsync(…).

LUHN checksum validation

The `badcardnumber` message is triggered when the card number doesn't pass the LUHN checksum. This can be tested with a card number of 1234561234561234.

Unknown Card Type validation

The `unknowncardtype` message is triggered when the card type cannot be recognized from the first six digits of the card number. This can be tested with 1234561234561238.

Note: The LUHN checksum is tested first. Unknown card type validation is tested only if the LUHN checksum passes. In both cases, the error is not reported until the user Submits the form. The message clears when the user begins typing in the Card Number field again.

Add a Card Without a Transaction

If you want to collect card information but without doing a transaction, you can set:

```
data-conligo-transactiontype="AddCard"
```

NOTE: ***that this causes data-conligo-show-card-number-list to be ignored, so no dropdown list will be shown.***

When the card information is entered into the Widget and submitted, the card information will be sent to the payment gateway to be vaulted (tokenized) but no transaction will be performed.

Like the 'Auth' and 'Sale' transaction types, a *responseHandler* function will handle the result of the 'AddCard' operation.

```
function responseHandler(response) {

    if (response.Result === "Success") {

        if (response.TrxType === "AddCard") {
            window.location.href = "ThankYouForAddingYourCard.aspx";
        }
```

Example Bootstrap 5.x Pay Button

```
<button id="conligoPayButton" type="button"
        data-conligo-cardsafeurl="@cardSafeUrl"
        data-conligo-trxcompleteurl="@Url.Action("TransactionComplete", "WebPayment", routeValues: new { orderNumber = orderNumber, sid = sid })"
        data-conligo-tenantid="@tenantId"
        data-conligo-accountselector="@currencyCode"
        data-conligo-customerid="@customerId"
        data-conligo-amount="@trxAmount"
        data-conligo-currencycode="@currencyCode"
        data-conligo-transactiontype="@transactionType"
        data-conligo-merchantreference="@orderNumber"
        data-conligo-title-html="Card Information"
        @*Address Information*@
        data-conligo-show-address-fields="false"
        data-conligo-address-line-1-value="@postalCode"
        data-conligo-address-city-value="@city"
        data-conligo-address-region-value="@region"
        data-conligo-address-country-value="@country"
        data-conligo-address-postal-code-value="@postalCode"
        @*Card and Button Labels*@
        data-conligo-card-number-label="Card Number"
        data-conligo-cardholder-label="Cardholder"
        data-conligo-card-expiration-label="MM/YY"
        data-conligo-security-code-label="CVV"
        data-conligo-close-button-label="Cancel"
        data-conligo-pay-button-label="Buy Now"
        @*Address Field Labels*@
        data-conligo-address-line-1-label="Address"
        data-conligo-address-city-label="City"
        data-conligo-address-region-label="Region"
        data-conligo-address-country-label="Country"
        data-conligo-address-postal-code-label="Postal Code"
        @*CSS Classes / Customizing the Layout*@
        data-conligo-css-form=""
        data-conligo-css-header-row="bg-dark text-light bg-gradient mt-0 py-2"
        data-conligo-css-row="form-floating mb-1"
        data-conligo-css-label=""
        data-conligo-css-input="form-control"
        data-conligo-css-select="form-select"
        data-conligo-css-button="btn"
        data-conligo-css-button-row="float-end"
        data-conligo-css-rule="hr-style-16"
        @*CSS Classes/ customizing Address fields and Card data fields*@
        data-conligo-css-card-number=""
        data-conligo-css-cardholder=""
        data-conligo-css-card-expiration=""
        data-conligo-css-security-code=""
        data-conligo-css-address-line-1=""
        data-conligo-css-address-city=""
        data-conligo-css-address-country=""
        data-conligo-css-address-region=""
        data-conligo-css-address-postal-code="w-50"
        @*CSS Classes/ customizing Address field Labels and Card data field Labels*@
        data-conligo-css-card-number-label="form-label"
        data-conligo-css-cardholder-label="form-label"
        data-conligo-css-card-expiration-label="form-label"
        data-conligo-css-security-code-label="form-label"
        data-conligo-css-address-line-1-label=""
        data-conligo-css-address-city-label=""
        data-conligo-css-address-country-label=""
        data-conligo-css-address-region-label=""
        data-conligo-css-address-postal-code-label=""
        @*CSS Classes/ Customizing Pay and Close Buttons*@
        data-conligo-css-pay-button="btn-primary"
        data-conligo-css-close-button="btn-danger"
        @*CSS Classes/ Customizing the Modal Dialog*@
        data-conligo-css-modal-sizing="modal-lg"
        data-conligo-css-modal-header=""
        data-conligo-css-modal-content="bg-light text-dark"
        data-conligo-css-modal-body=""
        data-conligo-responsehandler="responseHandler"
        class="btn btn-sm btn-success w-100 my-1 border d-block CardSafePayButton">
    <span>Pay</span>
</button>
```

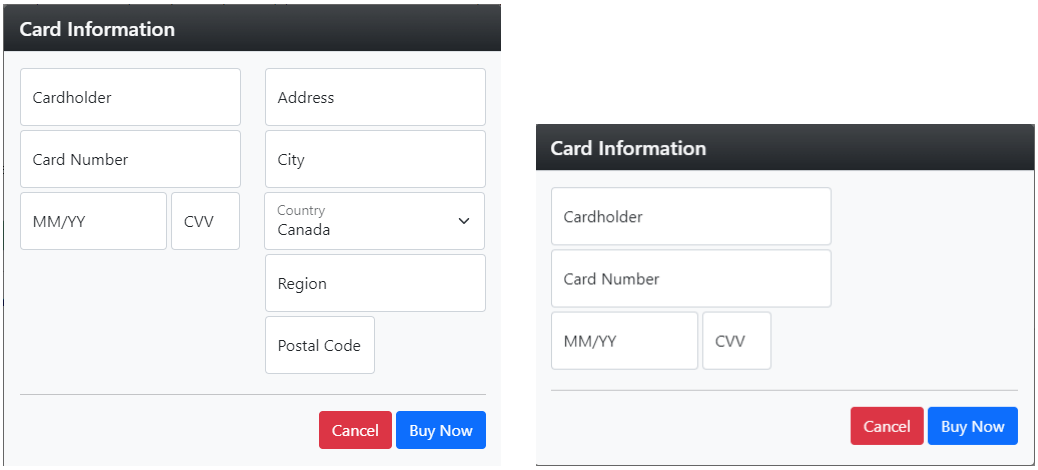

Figure 3 Example Bootstrap 5 rendering with and without address fields

Appendix A: Session Persistence when Challenged by 3D-Secure

3D-Secure determines that the cardholder must authenticate, a challenge form is displayed. This form comes from a different URL. So that it can return to your site, you pass it the return URL in data-conligo-trxcompleteurl. Upon the challenge form redirecting to the return URL, however, the session can be lost due to the setting of <sessionState cookieSameSite="xxx"/>.

For the tightest security, it is generally recommended that cookieSameSite be set to "Strict". This prevents the session cookie from being sent in an HTTP request if the page that issues the request is not in the same 'cookie domain'.

This behaviour causes a problem when dealing with a 3D-Secure challenge form because it results in the session being lost and a new session being creating upon return from the challenge form.

To remedy this, your return URL should include a parameter: sid=<%=Session.SessionID%>

Then, in Global.asax.cs, add this method:

```
 private void PreserveSessionId()
 {
    var sessionStateSection = ConfigurationManager.GetSection("system.web/sessionState")
                                  as SessionStateSection;
    string sessionCookieName = sessionStateSection != null &&
                                    !string.IsNullOrEmpty(sessionStateSection.CookieName)
           ? sessionStateSection.CookieName
           : "ASP.NET_SessionId";

    HttpCookie sessionCookie = Request.Cookies[sessionCookieName];
    if (sessionCookie == null)
    {
       // No session cookie found.
       // If we were sent one in the 'sid' parameter, make a cookie and add it to the request.
       // ASP.NET will connect to that session.
       // Otherwise, we'll let ASP.NET do its usual behaviour.
       var sid = Request.QueryString["sid"];
       if (!string.IsNullOrEmpty(sid))
       {
          Request.Cookies.Add(new HttpCookie(sessionCookieName, sid));
       }
    }
 }
```

Call this method from Application_BeginRequest(…):

```
    protected void Application_BeginRequest(object sender, EventArgs e)
    {
       PreserveSessionId();
    }
```

Appendix B: 3D-Secure Enrollment

The 3D-Secure enrollment response passed to the JavaScript responseHandler function looks like the following:

```
{
   string Result              // check for “Success”
   string TrxId               // links to the transaction state in Card Safe
   bool EnrollmentResult // true if card is enrolled in 3D-Secure
   bool MustChallenge     // true if the challenge form must be displayed
   string ChallengeForm   // the HTML form to be displayed to the cardholder
   string ErrorMessage
}
```

The PaymentStatusInfo object returned after processing the transaction is defined as follows:

```
{
   bool Approved
   string GatewayReference  // a unique value to identify the transaction at the payment gateway
   string ApprovalCode         // a secondary value to help identify the transaction,
                               // also called ‘authorization number’
   int ErrorCode               // this can be ignored as it may be different depending on the
                               // payment gateway in use.  ErrorDescription is more meaningful.
   string ErrorDescription

   AvsCvvResults avsCvvResults
}
```

The AvsCvvResults object returned in PaymentStatusInfo is defined as follows:

```
{
    /// ====================================================================
    /// <summary>
    /// The AVS result received from the bank.
    /// The bank AVS code should be tracked in the database for troubleshooting purposes.
    /// Be aware that the code found here is gateway/bank specific.
    /// </summary>
    string BankAvsCode

    /// ====================================================================
    /// <summary>
    /// The AVS message received from the bank.
    /// The bank AVS code should be tracked in the database for troubleshooting purposes.
    /// Be aware that the message found here is gateway/bank specific.
    /// </summary>
    string BankAvsMessage

    /// ====================================================================
    /// <summary>
    /// The bank's CVV match result.
    /// The bank CVV result code should be tracked in the database for troubleshooting purposes.
    /// Be aware that the code found here is gateway/bank specific.
    /// </summary>
    string BankCvvCode

    /// ====================================================================
    /// <summary>
    /// The bank's CVV message.
    /// The bank CVV code should be tracked in the database for troubleshooting purposes.
    /// Be aware that the message found here is gateway/bank specific.
    /// </summary>
    public virtual string BankCvvMessage { get; set; }

    /// ====================================================================
    /// <summary>
    /// Iciniti Card Safe AVS Code.
    /// This field contains a 'normalized' value.
    /// Gateways don't use all the same codes to indicate the same AVS/CVV results.
    /// Card Safe maps these different values to a normalized set of values.
    /// Possible values are:
    /// Unsupported    : AVS not supported
    /// NoMatch        : No match
    /// AddressMatched : Address match only
    /// ZipMatched     : Zip Match only
    /// FullMatch      : Address and Zip match
    /// </summary>
    IcinitiAvsResultEnum IcinitiAvsCode
```

```
    /// ===================================================================
    /// <summary>
    /// Iciniti CVV result code.
    /// This field contains a 'normalized' value.
    /// Gateways don't use all the same codes to indicate the same AVS/CVV results.
    /// Card Safe maps these different values to a normalized set of values.
    /// </summary>
    /// <return>One of NotSupported, NoMatch, Matched.</return>
    IcinitiCvvResultEnum IcinitiCvvCode
}
```