Successful integration of mobile payment via webview

Implementation guide

Document version 1.4

Contents

1. HISTORY OF THE DOCUMENT

Version	Author	Date	Comment
1.4	Lyra Network	10/16/2019	Renaming of the classes in code samples
1.3	Lyra Network	10/16/2018	Complete overhaul of the document
1.2	Lyra Network	3/16/2018	Adding the chapter **How can the native application detect the end of payment?** Updating the chapter **Sending the payment request from the merchant server**: adding the *vads_theme_config* parameter
1.1	Lyra Network	1/4/2018	Addition of chapter about optimizing performance
1.0	Lyra Network	11/10/2017	Initial version

2. PRESENTATION

PayZen offers you a unique solution for integrating mobile payment into your applications.

Our solution covers native iOS and Android applications. It is based on the use of the **webview** component.

A WebView allows to display content that is already available on the web within the application.

Thus, the PayZen solution for mobile payment via WebView offers several advantages to the merchant:

- A unique web and mobile configuration.

  You can copy the payment configuration of your website.

  Enabled payment methods, anti-fraud rules, etc. are included in the mobile application.

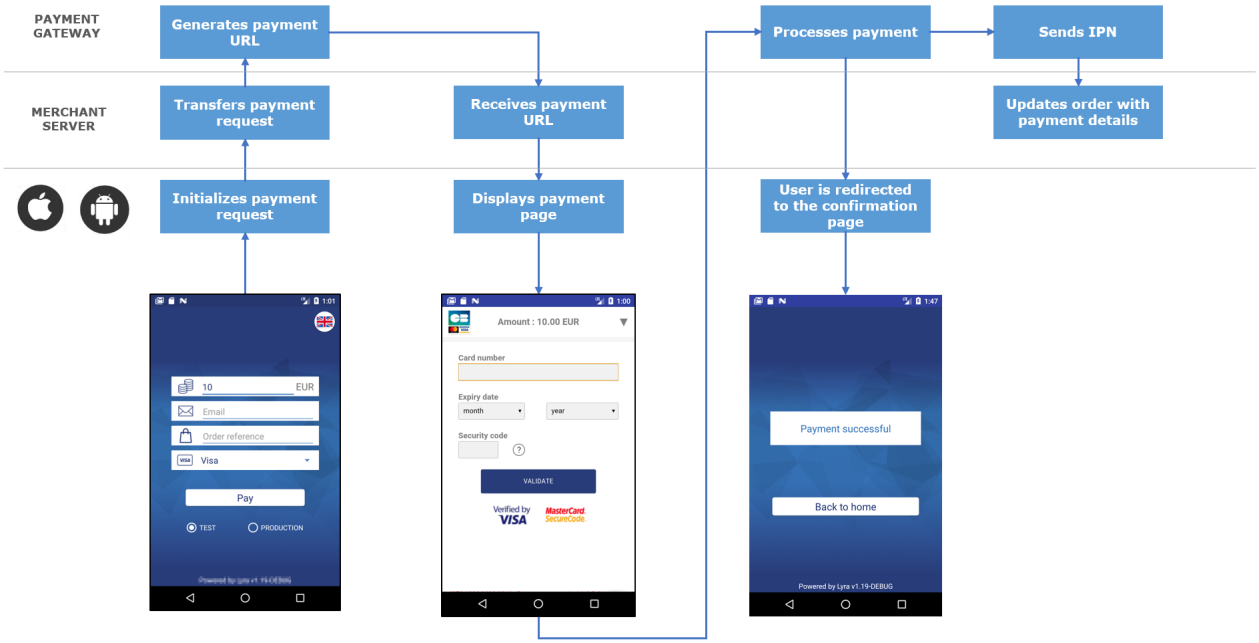
- Consistency in the display of information related to the buyer journey.

  Our payment pages are responsive and, therefore, are able to adapt to the different terminals of your customers (mobile, tablet or desktop).

- High security thanks to our PCI DSS certification on the one hand, and to the 3DS management integrated in the payment path on the other hand.

PCI DSS (= Payment Card Industry Data Security Standard) is the security standard of the payment card industry. It is a data security standard for major payment card networks such as Visa, MasterCard, American Express, Discover and JCB.

3. PAYMENT PROCESS



The buyer validates the shopping cart.

1. The mobile application initializes a payment request via the merchant server.
2. The merchant server sends a payment request to the gateway.
3. The gateway generates a payment URL and returns it to the mobile application.
4. The merchant server sends a payment URL to the mobile application.
5. The mobile application opens the payment page in a webview.
6. The buyer enters his or her card details and clicks **Validate**.
7. The gateway proceeds to payment, then transmits the payment notification to the merchant server.
8. The merchant server analyzes the payment result.
9. The buyer is automatically redirected to the merchant application.

4. PAYMENT INTEGRATION

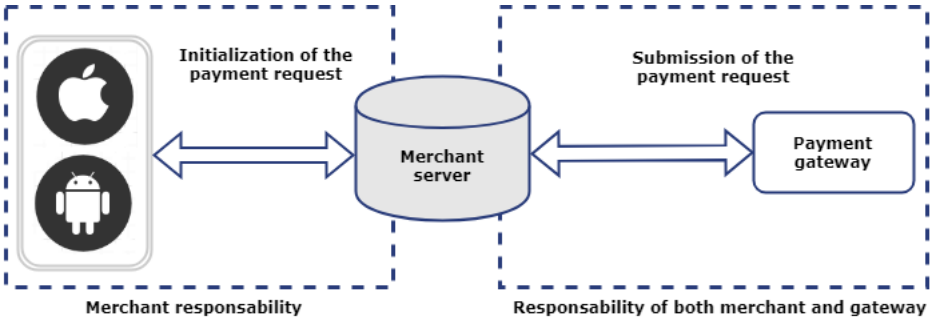
Code samples are provided to facilitate integration:

Merchant server *https://github.com/lyra/webview-payment-sparkjava-integration-sample*
iOS *https://github.com/lyra/webview-payment-ios-integration-sample*
Android *https://github.com/lyra/webview-payment-android-integration-sample*

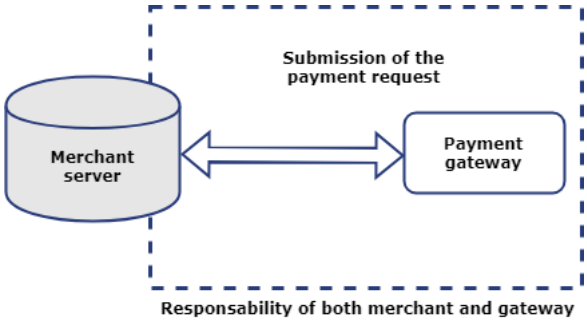
IMPORTANT
Make sure you read the comments present in the readme files before you start the application. The `MainActivity.kt` and `app-configuration.properties` files must be modified following the instructions provided in the comments.

The integration occurs in two phases:

- integration of data exchange between the merchant server and the payment gateway
- integration of data exchange between the mobile application and the merchant server.



5. PHASE 1: MERCHANT SERVER:



5.1. Transferring the payment request

The merchant server receives a payment request from the mobile application and must transmit it to the payment gateway.

To do this, the merchant website will generate an HTML payment form that it will post to the payment gateway.

The integrity of shared data is guaranteed by the exchange of alphanumeric signatures between the payment gateway and the merchant server.

The merchant server will transmit the alphanumeric signature in the payment form (see chapter **Computing the signature** of the **Hosted Payment page Implementation Guide** available in our online documentation archive).

IMPORTANT
All the data in the form must be encoded in UTF-8.
This will allow for the payment gateway to correctly interpret special characters (accents, punctuation marks, etc.).
Otherwise, the signature will be computed incorrectly and the form will be rejected.

1. Creation of the payment form

To create the payment form:

1. Use all the fields of the table below to build your payment request.

Field name	Description	Format	Value
vads_site_id	Shop ID	n8	E.g.: 12345678
vads_currency	Numeric currency code to be used for the payment, in compliance with the ISO 4217 standard (numeric code).	n3	E.g.: 978 for euro (EUR)
vads_amount	Payment amount in the smallest currency unit (cents for euro).	n..12	E.g.: 3000 for 30,00 EUR
vads_cust_email	Buyer's e-mail address	ans..150	E.g : abc@example.com
vads_payment_cards	Card type.	String	E.g.: **VISA** (See the Hosted Payment page Implementation Guide to view the list of possible values).
vads_order_id	Order ID	ans..64	E.g.: 2-XQ001
vads_version	Version of the exchange protocol with the payment gateway	enum	**V2**

Field name	Description	Format	Value
vads_theme_config	Allows to improve performance by deactivating some elements of the payment page, such as the language selector, the logos at the bottom of the page, etc.	map	**SIMPLIFIED_DISPLAY=true**
vads_trans_date	Date and time of the payment form in UTC format	n14	Respect the YYYYMMDDhhmmss format E.g.: 20170701130025
vads_trans_id	Transaction number	n6	E.g.: 123456
vads_payment_config	Payment type	enum	**SINGLE** for immediate payment **MULTI** for installment payment
vads_page_action	Action to perform	enum	**PAYMENT**
vads_ctx_mode	Defines the mode of interaction with the payment gateway.	enum	**TEST** or **PRODUCTION**
vads_action_mode	Acquisition mode for credit card data	enum	**INTERACTIVE**
signature	Signature guaranteeing the integrity of the requests exchanged between the merchant website and the payment gateway.	ans44	E.g.: NrHSHyBBBc +TtcauudspNHQ5cYcy4tS4IjvdC0ztFe8=

2. Use the fields below to manage the return to the mobile application at the end of the payment.

   A payment can result in 4 different states:

   - Payment accepted,
   - Payment declined,
   - Payment error,
   - Payment abandoned by the buyer.

   You must associate a URL to each status:

Field name	Description	Format	Value
vads_url_success	URL where the buyer will be redirected in case of a **successful transaction**.	ans..1024	E.g.: **http://webview.success**
vads_url_refused	URL where the buyer will be redirected in case of a **declined transaction**.	ans..1024	E.g.: **http://webview.refused**
vads_url_cancel	URL where the buyer will be redirected in case of an **abandoned or expired transaction** (timeout).	ans..1024	E.g.: **http://webview.cancel**
vads_url_error	URL where the buyer will be redirected in case of an **error**.	ans..1024	E.g.: **http://webview.error**

3. Use the fields below to configure the time of redirection to the mobile application at the end of the payment:

Field name	Description	Format
vads_redirect_success_timeout	Defines the delay before the redirection that follows an accepted payment. This delay is presented in seconds and must be between 0 and 300 sec. **Set this field to "0" to not display the payment ticket and to automatically redirect the buyer to the mobile application.**	n..3
vads_redirect_error_timeout	Defines the delay before the redirection that follows a declined payment. This delay is presented in seconds and must be between 0 and 300 sec.	n..3

Field name	Description	Format
	Set this field to "0" to not display the payment rejection page and to automatically redirect the buyer to the mobile application.	

Table 1: List of available optional fields

4. Add other optional fields depending on your requirements (see chapter **Using additional features** of the **Hosted Payment Page Implementation Guide** available in our online documentation archive).

5. See the chapter **Computing the signature** or the **Hosted Payment Page Implementation Guide** and compute the value of the **signature** field.

2. Sending the payment request

The payment creation API is available via POST at this address:

https://secure.payzen.eu/vads-payment/entry.silentInit.a

IMPORTANT
The URL of the payment creation API is different from the payment page URL, as described in the Hosted Payment Page Implementation Guide.

Excerpt from the sample code:

```
List<NameValuePair> formParameters = new ArrayList<>();

formParameters.add(new BasicNameValuePair("vads_action_mode", "INTERACTIVE"));
formParameters.add(new BasicNameValuePair("vads_amount", amount));
formParameters.add(new BasicNameValuePair("vads_ctx_mode", mode));
formParameters.add(new BasicNameValuePair("vads_currency", currency));
if (StringUtils.isNotEmpty(email)) {
formParameters.add(new BasicNameValuePair("vads_cust_email", email));
}

formParameters.add(new BasicNameValuePair("vads_language", language));
if (StringUtils.isNotEmpty(orderId)) {
 formParameters.add(new BasicNameValuePair("vads_order_id", orderId));
}
formParameters.add(new BasicNameValuePair("vads_page_action", "PAYMENT"));

//Set the card type if provided
if (StringUtils.isNotEmpty(cardType)) {
 formParameters.add(new BasicNameValuePair("vads_payment_cards", cardType.toUpperCase()));
}
formParameters.add(new BasicNameValuePair("vads_payment_config", "SINGLE"));
formParameters.add(new BasicNameValuePair("vads_site_id", merchantSiteId));
formParameters.add(new BasicNameValuePair("vads_theme_config", "SIMPLIFIED_DISPLAY=true"));
formParameters.add(new BasicNameValuePair("vads_trans_date",
 calculateDateFormatInUTC("yyyyMMddHHmmss")));
formParameters.add(new BasicNameValuePair("vads_trans_id", String.format("%06d",
 transactionId)));
formParameters.add(new BasicNameValuePair("vads_url_cancel", "http://webview_" +
 merchantSiteId + ".cancel"));
formParameters.add(new BasicNameValuePair("vads_url_error", "http://webview_" +
 merchantSiteId + ".error"));
formParameters.add(new BasicNameValuePair("vads_url_refused", "http://webview_" +
 merchantSiteId + ".refused"));
formParameters.add(new BasicNameValuePair("vads_url_return", "http://webview_" +
 merchantSiteId + ".return"));
formParameters.add(new BasicNameValuePair("vads_url_success", "http://webview_" +
 merchantSiteId + ".success"));
formParameters.add(new BasicNameValuePair("vads_version", "V2"));

//Create the string to sign
String concatenateMapParams = "";
for (NameValuePair pair : formParameters) {
 concatenateMapParams += pair.getValue() + "+";
}
//Add private key in signature
concatenateMapParams += usedMerchantKey;

//Add signature to form parameters
```



```
formParameters.add(new BasicNameValuePair("signature", hmacSha256(concatenateMapParams,
 usedMerchantKey)));
```


5.2. Receiving the Payment URL

The payment gateway returns a response in JSON format containing an HTTP success or error status code.

Success

In case of success, the payment gateway returns an HTTP status code `200 OK`.

The response contains the payment URL where the mobile application must redirect the buyer.

```
{
 "status":"INITIALIZED",
 "redirect_url":"https://secure.payzen.eu:443/vads-payment/
exec.refresh.a;jsessionid=CE2Cb9daEDe7f6dBF31FE65e.vadpayment01bdx"
}
```

Error

In case of error, the payment gateway returns an HTTP status code `400 Bad Request` or `500 Internal Server Error`.

The response will contain the details of the error.

```
{
 "status":"ERROR",
 "error":" {"code": "09", "value": "Missing or invalid parameter value"}"
}
```

For more details, see the list of error codes of the Hosted Payment Page Implementation Guide:

https://payzen.io/de-DE/error-code/error-00.html

Excerpt from the sample code:

```
//If the HTTP return code is 200 (OK) we prepare the generated URL
if (httpResponseCode == 200) {
 if ("INITIALIZED".equals(responseData.get("status"))) {
  redirectionUrl = responseData.get("redirect_url");
 } else {
  //Payment could  not be created. Maybe a missing parameter, an invalid value or signature?
  //Use logs here in order to detect and fix the real cause
  throw new RuntimeException("Error in payment initialization. Returned error: " +
 responseData.get("error"));
 }
} else {
 throw new RuntimeException("Error in payment initialization. HTTP errorCode: " +
 httpResponse);
}
```

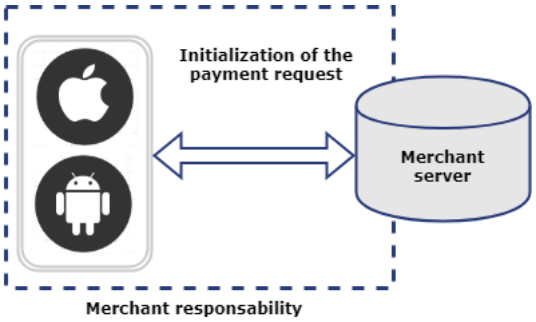
5.3. Processing the notification at the end of payment (IPN)

Once the payment has been made, the payment gateway notifies the merchant server about the transaction result.

The data will be sent with the notification URL defined in the .

See the **Hosted Payment Page Implementation Guide** available in our online documentation archive for information on configuring the notification rules and analyzing the transmitted data.

6. PHASE 2: MOBILE APPLICATION



6.1. Initializing the payment request

When the buyer validates his or her order, the application generates a "payload" containing the details of the shopping cart, the buyer contact information, the shipping details, etc...

The provided example uses the following data:

```
{
 "email": "example@email.com",
 "orderId": "myOrderId-1",
 "amount": "200",
 "currency": "978",
 "mode": "TEST",
 "language": "de",
 "cardType": ""
}
```

The mobile application transmits the payment request to the merchant server via a POST request method.

Excerpt from the sample code for Android:

```
val conn = URL(serverUrl).openConnection() as HttpURLConnection
conn.requestMethod = "POST"
conn.setRequestProperty("Content-type", "application/json")
conn.setRequestProperty("Accept", "*/*")
conn.doInput = true
conn.doOutput = true
conn.connectTimeout = 15000

val os = conn.outputStream
val writer = BufferedWriter(OutputStreamWriter(os, "UTF-8"))
writer.write(payload.toString())
writer.flush()
writer.close()
os.close()

conn.connect()

val out = OutputStreamWriter(conn.outputStream)
```


Excerpt from the sample code for iOS:

```
/// Build an URLRequest according required payment information : server url, email, amount,
 mode, lang
///
/// - Returns: URLRequest object
func buildRequest() -> URLRequest? {
 let serverUrl: NSURL = NSURL(string: PaymentProvider.SERVER_URL)!
 var urlRequest = URLRequest(url:serverUrl as URL)
 urlRequest.httpMethod = "POST"
 var params: [String: String] = ["amount": paymentInfo.amount, "currency":
 paymentInfo.currency, "mode": paymentInfo.mode, "language": paymentInfo.lang]
 if !paymentInfo.email.isEmpty{
  params["email"] = paymentInfo.email
 }
 if !paymentInfo.cardType.isEmpty{
  params["cardType"] = paymentInfo.cardType
 }
 do{
  let jsonParam = try JSONSerialization.data(withJSONObject: params, options: [])
  urlRequest.httpBody = jsonParam
 }
 catch{
  return nil
 }

 return urlRequest
}
```

6.2. Displaying the payment page in a web view

Once the request is processed, the payment gateway returns the payment URL to the mobile application.

The application initializes a webview and displays a payment page.

Exerpt of code sample for Android (**Kotlin**)

```
val webView = WebView(this)

// Url loading
webView.loadUrl(url)

// Enable javascript
webView.settings.javaScriptEnabled = true

// To allow debug WebView from Chrome Dev Tools
WebView.setWebContentsDebuggingEnabled(false)

// Define new web view client by overriding shouldOverrideUrlLoading method in order to check
 urls
webView.webViewClient = object: WebViewClient() {
 override fun onPageFinished(view: WebView, url: String) {
  progressBar.visibility = View.GONE
  super.onPageFinished(view, url)
 }

 @Suppress("OverridingDeprecatedMember")
 override fun shouldOverrideUrlLoading(view: WebView, url: String): Boolean {
  return checkUrl(webView, url)
 }

 @TargetApi(Build.VERSION_CODES.LOLLIPOP)
 override fun shouldOverrideUrlLoading(view: WebView, webResourceRequest: WebResourceRequest):
 Boolean {
  return checkUrl(webView, webResourceRequest.url.toString())
 }
}
webView.canGoForward()
```


Exerpt of code sample for iOS (**Swift**)

```
/// Call server to get payment url, supply a block completion (callback)
///
/// - Returns: status boolean, payment url
func getPaymentContext(completion: @escaping (Bool, String, NSError?) -> ()){
 // Build request
 let urlRequest = buildRequest()
 // Call server to obtain a payment Url
 // Completion is a callback, giving call status, and payment url if success
 if let request = urlRequest{
  let task = URLSession.shared.dataTask(with: request) { (data: Data?, response: URLResponse?,
 error: Error?) in
   if error != nil{
    completion(false, "", NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_NO_CONNECTION.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_NO_CONNECTION.errorMsg]))
   }
   if let httpResponse = response as? HTTPURLResponse {
    let json = try? JSONSerialization.jsonObject(with: data!, options: .mutableContainers) as?
 NSDictionary
    var redirectionUrl = ""
    var errorMsg = ""
    if let jsonResponse = json {
     redirectionUrl = (jsonResponse!["redirectionUrl"] as? String)!
     errorMsg = (jsonResponse!["errorMessage"] as? String)!
    }
    switch(httpResponse.statusCode){
     case 200:
      completion(true, redirectionUrl, nil)
     case 400, 500:
      completion(false, "", NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_SERVER.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_SERVER.errorMsg + errorMsg]) )
     default:
      completion(false, "", NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_UNKNOW.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_UNKNOW.errorMsg]))
    }
   }
   else{
    completion(false, "", NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_TIMEOUT.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_TIMEOUT.errorMsg]))
   }
 }
 task.resume()
 } else{
  completion(false, "", NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_UNKNOW.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_UNKNOW.errorMsg]))
 }
}
```

6.3. Detection of the end of the payment

In order to detect the end of payment, it is necessary to analyze the different URLs that go through the webview.

Depending on the URL, the mobile application can:

- accept the propagation of the URL and display the page in the webview
  (for example, during a payment request, allow the buyer to make his or her payment)
- refuse the propagation of the URL and regain control.
  (for example, in case of success and at the end of payment, allow the buyer to return to the native application)

Thanks to the listening mechanism of URLs, you can control the payment progress and decide when to switch to your native application.

Exerpt of code sample for Android (Kotlin)

```
private fun checkUrl(view: WebView, url: String): Boolean {
 val isCallBack = isCallbackUrl(url)

 when {
  // payment is finish
  isCallBack -> {
   view.stopLoading()
   goToFinalActivity(url)
  }
  else -> view.loadUrl(url)
 }
 return (!isCallBack)
}
```

Exerpt of code sample for iOS (**Swift**) :

```
func notifyPaymentFinish(navigationAction: WKNavigationAction){
 let webViewUrlResponse = self.buildWebviewUrlResponse(navigationAction: navigationAction)
 var error: NSError?
 switch webViewUrlResponse.paymentStatus {
  case "success":
   error = nil
  case "cancel":
   error = NSError.init(domain: PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_PAYMENT_CANCELATION.errorCode, userInfo:
 [NSLocalizedFailureReasonErrorKey: PaymentProvider.ERROR_PAYMENT_CANCELATION.errorMsg])
  case "refused":
   error = NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_PAYMENT_REFUSED.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_PAYMENT_REFUSED.errorMsg])
  default:
   error = NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_UNKNOW.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_UNKNOW.errorMsg])

 }
 self.dismiss(animated: true) {
  self.paymentDelegate?.didPaymentProcessFinish(error: error)
 }
}
```