COLLECTING SOLUTION

Successful integration of mobile payment via webview

Implementation Guide

Document version 1.5

Contents

1. HISTORY OF THE DOCUMENT

Version	Author	Date	Comment
1.5	Lyra Collect	12/05/2020	• Server-side integration: addition of form fields for transmitting oder, buyer and shipping details. • Addition of signature computation. • Integration redesign on the mobile application side. • Addition of camera and NFC card scanning support.
1.4	Lyra Collect	16/10/2019	Initial version

2. PRESENTATION

Lyra Collect 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.

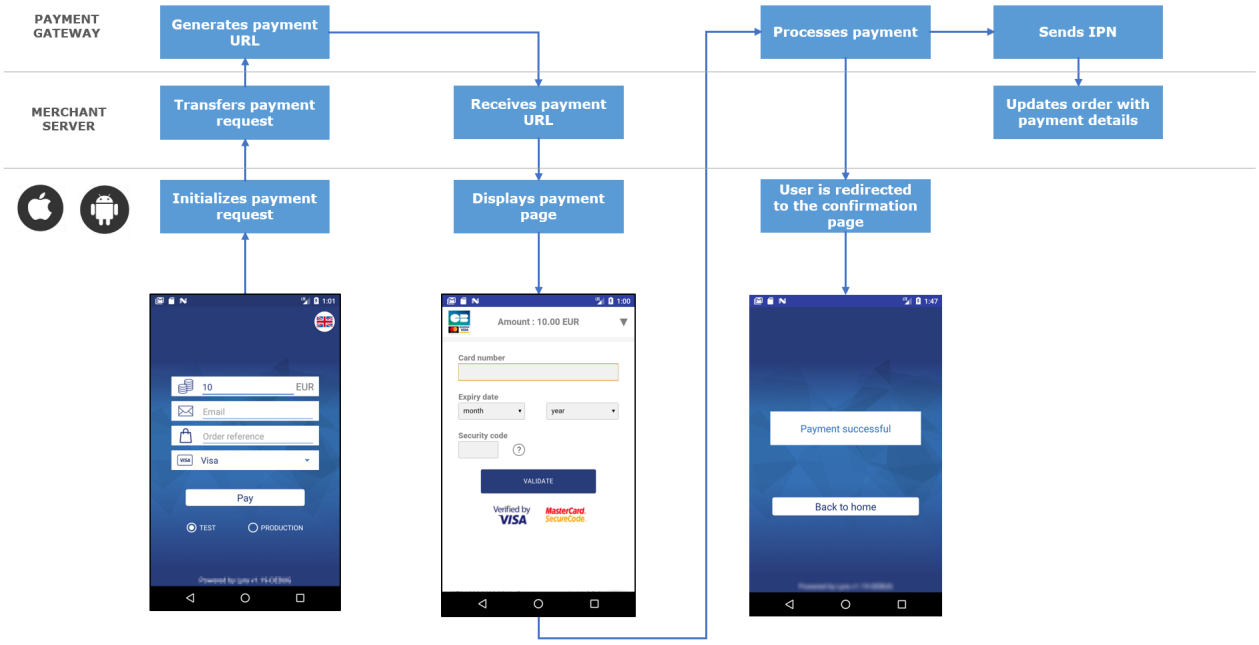
A WebView allows to display content that is already available on the web within the application.

Thus, the Lyra Collect 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

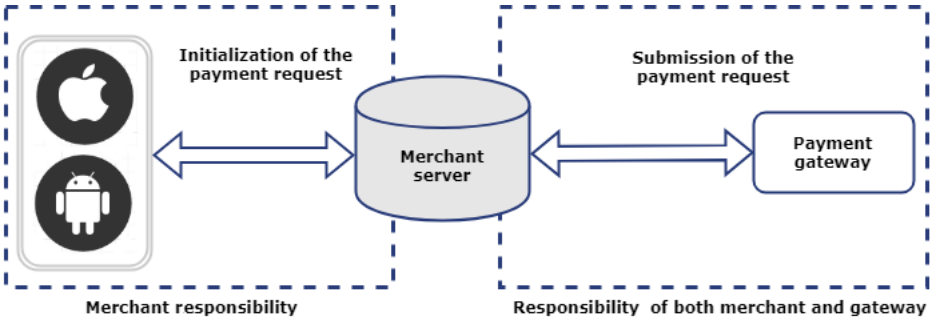
Code samples are provided to facilitate integration:

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*

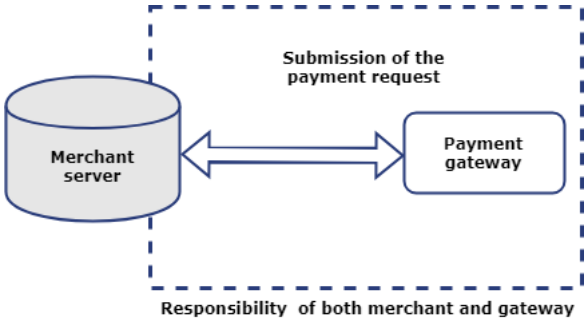
IMPORTANT
Make sure you read the comments present in the readme files before you start the application. The `MainActivity.kt` and `app-configuration.properties` files must be modified following the instructions provided in the comments.

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. Creation of the payment form

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 on page 14).

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.

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 *Implementation guide - Hosted Payment Form* 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**
vads_theme_config	Allows to improve performance by disabling some elements of the payment page, such as the language selector, the logos at the bottom of the page, etc.	map	**SIMPLIFIED_DISPLAY=true**

Field name	Description	Format	Value
vads_trans_date	Date and time of the payment form in UTC format	n14	Respect the YYYYMMDDHHMMSS format E.g.: 20200101130025
vads_trans_id	Transaction number	an6	E.g.: xrT15p
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 payment method data	enum	**INTERACTIVE**
signature	Signature guaranteeing the requests integrity exchanged between the merchant website and the payment gateway. Its value calculation is described here: Computing the signature on page 14.	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.	

4. Add other optional fields according to your needs (see the following subsections).

Transmitting buyer details

The Merchant can specify the buyer's billing details (e-mail address, title, phone number, etc.). This information will be used to create the invoice.

All the data transmitted via the payment form can be viewed in the transaction details in the Expert Back Office (**Buyer** tab).

Use optional fields according to your requirements. *These fields will be returned with the response and will include the value transmitted in the form.*

Field name	Description	Format	Value
vads_cust_email	Buyer's e-mail address	ans..150	E.g.: abc@example.com
vads_cust_id	Buyer reference on the merchant website	an..63	E.g.: C2383333540
vads_cust_title	Buyer's title	an..63	E.g.: M.
vads_cust_status	Status	enum	**PRIVATE**: for a private individual **COMPANY**: for a company
vads_cust_first_name	First name	ans..63	E.g.: Laurent
vads_cust_last_name	Name	ans..63	E.g.: Durant
vads_cust_legal_name	Buyer's legal name	an..100	E.g.: D. & Cie
vads_cust_cell_phone	Cell phone number	an..32	E.g.: 06 12 34 56 78
vads_cust_address_number	Street number	ans..64	E.g.: 109
vads_cust_address	Postal address	ans..255	E.g.: Rue de l'innovation
vads_cust_address2	Second line of the address	ans..255	E.g.:
vads_cust_district	District	ans..127	E.g.: Downtown
vads_cust_zip	Zip code	an..64	E.g.: 31670
vads_cust_city	City	an..128	E.g.: Labège
vads_cust_state	State / Region	ans..127	E.g.: Occitanie
vads_cust_country	Country code in compliance with the ISO 3166 standard	a2	E.g.: "FR" for France, "PF" for French Polynesia, "NC" for New Caledonia, "US" for the United States

Note

vads_cust_phone and **vads_cust_cell_phone** fields support all formats:

Examples:

- 0123456789
- +33123456789
- 0033123456789
- (00.571) 638.14.00
- 40 41 42 42

Transmitting order details

The merchant can transmit the order details (order reference, description, shopping cart content, etc.).

This information can be found in the transaction details in the Expert Back Office.

1. Use optional fields according to your requirements. These fields will be returned in the response and will include the value transmitted in the form.

Field name	Description	Format	Value
vads_order_info	Complementary order info	an..255	E.g.: Door phone code 3125
vads_order_info2	Complementary order info	an..255	E.g.: No elevator
vads_order_info3	Complementary order info	an..255	E.g.: Express
vads_nb_products	Number of items in the cart	n..12	E.g.: 2
vads_product_ext_idN	Product bar code on the merchant website. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).		E.g.: vads_product_ext_id0 = "0123654789123654789" vads_product_ext_id1 = "0223654789123654789"
vads_product_labelN	Item name. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	ans..255	E.g.: vads_product_label0 = "Dated 3 days stay" vads_product_label1 = "Private concert"
vads_product_amountN	Item amount expressed in the smallest currency unit. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	n..12	E.g.: vads_product_amount0 = "32150" vads_product_amount1 = "10700"
vads_product_typeN	Item type. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	enum	E.g.: vads_product_type0 = "TRAVEL" vads_product_type1 = "ENTERTAINMENT"
vads_product_refN	Item reference. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	an..64	E.g.: vads_product_ref0 = "1002127784" vads_product_ref1 = "1002127693"
vads_product_qtyN	Quantity of items. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	n..12	E.g.: vads_product_qty0 = "1" vads_product_qty1 = "1"

2. Populate the **vads_nb_products** field with the number of items contained in the cart.

   Note:

   This field becomes mandatory for the shopping cart to be taken into account.

   *When it is populated, the **Shopping cart** tab becomes available in the transaction details in the Expert Back Office.*

   *However, if the other fields that start with **vads_product_** are not populated, the tab will not include any information. For this reason, when populating the **vads_nb_products** field, it becomes mandatory to populate the other fields that start with **vads_product_**.*

3. Populate the **vads_product_amountN** field with the amount for the items in the cart, using the smallest currency unit.

   N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).

4. Populate **vads_product_typeN** with the value corresponding to the item type.

   N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).

Value	Description
FOOD_AND_GROCERY	Food and grocery

Value	Description
AUTOMOTIVE	Cars / Moto
ENTERTAINMENT	Entertainment / Culture
HOME_AND_GARDEN	Home and gardening
HOME_APPLIANCE	Household appliances
AUCTION_AND_GROUP_BUYING	Auctions and group purchasing
FLOWERS_AND_GIFTS	Flowers and presents
COMPUTER_AND_SOFTWARE	Computers and software
HEALTH_AND_BEAUTY	Health and beauty
SERVICE_FOR_INDIVIDUAL	Services for individuals
SERVICE_FOR_BUSINESS	Services for companies
SPORTS	Sports
CLOTHING_AND_ACCESSORIES	Clothes and accessories
TRAVEL	Travel
HOME_AUDIO_PHOTO_VIDEO	Sound, image and video
TELEPHONY	Telephony

5. Populate **vads_product_labelN** with the name of each item contained in the cart.
   N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).
6. Populate **vads_product_qtyN** with the quantity of each item contained in the cart.
   N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).
7. Populate **vads_product_refN** with the reference of each item contained in the cart.
   N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).
8. Check the value of the **vads_amount** field. It must correspond to the total amount of the order.

Transmitting shipping details

The merchant can transmit the buyer's shipping details (e-mail address, title, phone number etc.).

This information can be found in the transaction details in the Expert Back Office (**Delivery** tab).

Use optional fields according to your requirements. *These fields will be returned in the response and will include the value transmitted in the form.*

Field name	Description	Format	Value
vads_ship_to_city	City	an..128	E.g.: Bordeaux
vads_ship_to_country	Country code in compliance with the ISO 3166 standard (required for triggering one or more actions if the **Shipping country control** profile is enabled).	a2	E.g.: FR
vads_ship_to_district	District	ans..127	E.g.: La Bastide
vads_ship_to_first_name	First name	ans..63	E.g.: Albert
vads_ship_to_last_name	Name	ans..63	E.g.: Durant
vads_ship_to_legal_name	Legal name	an..100	E.g.: D. & Cie
vads_ship_to_phone_num	Phone number	ans..32	E.g.: 0460030288
vads_ship_to_state	State / Region	ans..127	E.g.: Nouvelle Aquitaine
vads_ship_to_status	Allows to specify the type of the shipping address.	enum	**PRIVATE**: for shipping to a private individual **COMPANY**: for shipping to a company
vads_ship_to_street_number	Street number	ans..64	E.g.: 2
vads_ship_to_street	Postal address	ans..255	E.g.: Rue Sainte Catherine
vads_ship_to_street2	Second line of the address	ans..255	
vads_ship_to_zip	Zip code	an..64	E.g.: 33000

5.2. Computing the signature

To be able to compute the value of the **signature** field, you must have:

- all the fields that start with **vads_**
- the signature algorithm chosen in the shop configuration
- the **key**

The value of the key is available in your Expert Back Office via **Settings** > **Shop** > **Keys** tab.

The signature algorithm is defined in your Expert Back Office via **Settings** > **Shop** > **Configuration** tab.

For maximum security, it is recommended to use HMAC-SHA-256 algorithm and an alphanumeric key.

The use of SHA-1 algorithm is deprecated but maintained for compliance reasons.

Warning: you must not use the REST API keys for computing the signature of your payment form.

Only the use of the HMAC-SHA-256 algorithm is implemented in our code sample.

To compute the signature:

1. Sort the fields that start with **vads_** alphabetically.
2. Make sure that all the fields are encoded in UTF-8.
3. Concatenate the values of these fields separating them with the "**+**" character.
4. Concatenate the result with the test or production key separating them with a "**+**".
5. According to the signature algorithm defined in your shop configuration:
   a. if your shop is configured to use "SHA-1", apply the **SHA-1** hash function to the chain obtained during the previous step. **Deprecated.**
   b. if your shop is configured to use "HMAC-SHA-256", compute and encode in Base64 format the message signature using the **HMAC-SHA-256** algorithm with the following parameters:
      - the SHA-256 hash function,
      - the test or production key (depending on the value of the **vads_ctx_mode** field) as a shared key,
      - the result of the previous step as the message to authenticate.
6. Save the result of the previous step in the **signature** field.

Example of parameters sent to the payment gateway:

```
<form method="POST" action="https://secure.lyra.com/vads-payment/entry.silentInit.a">
<input type="hidden" name="vads_action_mode" value="INTERACTIVE" />
<input type="hidden" name="vads_amount" value="5124" />
<input type="hidden" name="vads_ctx_mode" value="TEST" />
<input type="hidden" name="vads_currency" value="978" />
<input type="hidden" name="vads_page_action" value="PAYMENT" />
<input type="hidden" name="vads_payment_config" value="SINGLE" />
<input type="hidden" name="vads_site_id" value="12345678" />
<input type="hidden" name="vads_trans_date" value="20170129130025" />
<input type="hidden" name="vads_trans_id" value="123456" />
<input type="hidden" name="vads_version" value="V2" />
<input type="hidden" name="signature" value="ycA5Do5tNvsnKdc/eP1bj2xa19z9q3iWPy9/rpesfS0="/>

<input type="submit" name="pay" value="Pay"/>
</form>
```

This sample form is analyzed as follows:

1. The fields whose names start with **vads_** are sorted **alphabetically**:
   - vads_action_mode
   - vads_amount
   - vads_ctx_mode
   - vads_currency
   - vads_page_action
   - vads_payment_config
   - vads_site_id
   - vads_trans_date
   - vads_trans_id
   - vads_version
2. The values of these fields are concatenated using the "**+**" character:

   ```
   INTERACTIVE+5124+TEST+978+PAYMENT+SINGLE+12345678+20170129130025+123456+V2
   ```

3. The value of the test key is added at the end of the chain and separated with the "**+**" character. In this example, the test key is **1122334455667788**

   ```
   INTERACTIVE+5124+TEST+978+PAYMENT+SINGLE+12345678+20170129130025+123456+V2+1122334455667788
   ```

4. If you use the SHA-1 algorithm, apply it to the obtained chain.

   The result that must be transmitted in the signature field is: **59c96b34c74b9375c332b0b6a32e6deeec87de2b**

5. If your shop is configured to use "HMAC-SHA-256", compute and encode in Base64 format the message signature using the **HMAC-SHA-256** algorithm with the following parameters:
   - the SHA-256 hash function,
   - the test or production key (depending on the value of the **vads_ctx_mode** field) as a shared key,
   - the result of the previous step as the message to authenticate.

   The result that must be transmitted in the signature field is:

   ycA5Do5tNvsnKdc/eP1bj2xa19z9q3iWPy9/rpesfS0=

5.3. Transferring the payment request

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

https://secure.lyra.com/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*.

5.4. 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.lyra.com: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://lyra.com/doc/en/collect/error-code/error-00.html

5.5. 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 Expert Back Office.

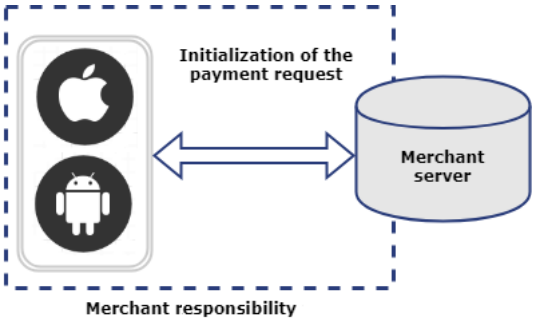
See the *Hosted Payment Page Implementation Guide* for further information on configuring the notification rules and analyzing the transmitted data.

5.6. Code sample

See the complete example for deploying the merchant server:

https://github.com/lyra/webview-payment-sparkjava-integration-sample

6. PHASE 2: MOBILE APPLICATION



Your mobile application integration must follow the following steps:

1. **Payment request initialization and transmission of required data**

   The application generates a “payload” containing the shopping cart data, buyer’s contact information, delivery details and transmits the payment request to the merchant server via a POST request.

2. **Payment page display in a webview**,

   The application initializes a webview and displays the payment page using the URL returned by the payment gateway.

3. **End of payment detection**.

   The mobile application must analyze the different URLs that pass through the webview. Since the return URLs are defined by the merchant server, you have control over the payment process and can decide when to switch over to your native application.

Integration details:

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

.

6.1. Camera card scanning

A cell phone camera can be used for pre-filling card details during the payment.

A complete integration example is provided here:

iOS *https://github.com/lyra/webview-payment-iOS-integration-sample/tree/card_scanning*
Android *https://github.com/lyra/webview-payment-android-integration-sample/tree/card_scanning*

This example uses external libraries developed by third-party providers.

IMPORTANT
Lyra Collect does not guarantee and is not responsible of the quality of external libraries.
The use of these libraries is not compatible with PCI-DSS.

6.2. NFC card scanning

NFC cell phone module can be used for pre-filling card details during the payment.

A complete integration example is provided here:

iOS Not available.

Android *https://github.com/lyra/webview-payment-android-integration-sample/tree/card_scanning_by_nfc*

This example uses external libraries developed by third-party providers.

IMPORTANT
Lyra Collect does not guarantee and is not responsible of the quality of external libraries.
The use of these libraries is not compatible with PCI-DSS.

