Integrating with Google Pay API for Mobile Payments

Document version 1.3

Contents

1. HISTORY OF THE DOCUMENT

Version	Author	Date	Comment
1.3	Lyra Network	25/04/2019	Indication of the Trade Mark label
1.2	Lyra Network	14/01/2019	• Update of the prerequisites • Addition of the MID creation process
1.1	Lyra Network	19/10/2018	• Update to the latest version of the Google Pay™ API • Update of code samples
1.0	Lyra Network	10/07/2018	Initial version

2. GETTING IN TOUCH WITH TECHNICAL SUPPORT

Looking for help? Check our FAQ on our website

https://payzen.io/fr-FR/faq/sitemap.html

For technical inquiries or support, you can reach us from Monday to Friday, between 9am and 6pm

by phone at:	0811708709 Service fee 0.06 € / mi + call charge
by e-mail:	*support@payzen.eu*
via your Merchant Back Office:	(Menu: **Help** > **Contact support**

To facilitate the processing of your demands, you will be asked to communicate your shop ID (an 8-digit number) .

This information is available in the "registration of your shop" e-mail or in the Merchant Back Office (**Settings** > **Shop** > **Configuration**).

3. PRESENTATION

Google Pay™ is a payment method allowing buyers to easily pay for their purchases using the payment card associated with their Google account.

It is a simple, fast and secure method of performing purchases online or using a mobile app.

There are two types of integration:

- mobile app integration
- integration on a merchant website (payment is made via buyer's browser)

This document describes mobile app integration using the payload returned by the Google Pay™ API. However, it is possible to adapt the following instructions by using a client payload for the Google Pay™ JavaScript API.

Google Pay™ express checkout:

During a payment, the buyer must enter their billing and shipping address in the mobile app.

In order to facilitate buyer's payments on different sites or apps, it is possible to manage billing and shipping addresses via Google wallet.

The buyer selects his or her payment method and addresses once and the merchant website exploits the data contained in the response (payload).

(The information is automatically transmitted to the payment gateway and is visible in the PayZen Back Office).

Important:

- The current implementation does not allow 3D Secure authentication for Google Pay™ payments. Therefore, there is no liability shift.
- The current implementation does not allow to create recurring payments (payments by subscription) with the Google Pay™ payment method.

4. PREREQUISITES

For the merchant:

- Opt for the PayZen Premium or PayZen Expert offer

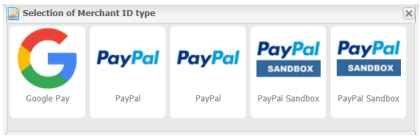
For the buyer:

- Have a Google account
- Associate a payment method with their Google account
- Have mobile equipment that is compatible with Google Pay services

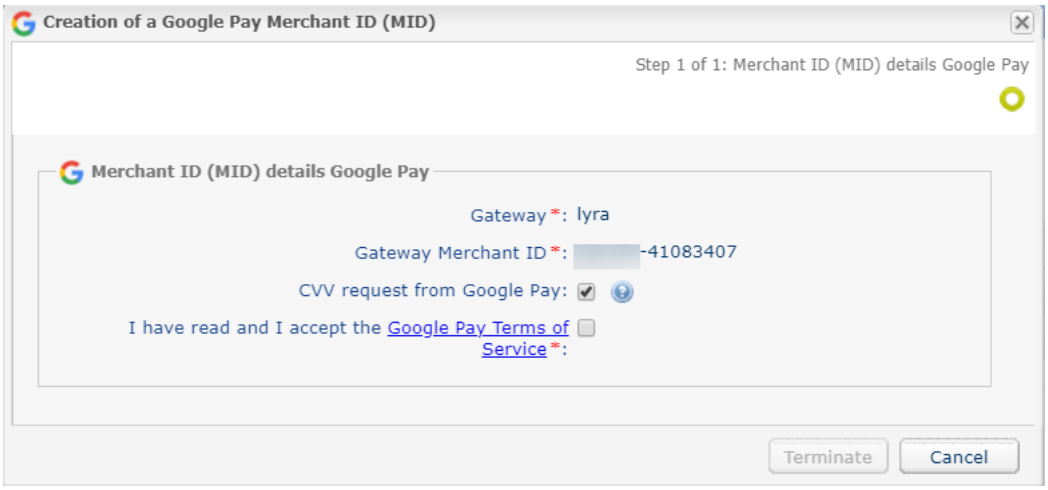
5. DECLARING A GOOGLE PAY™ MID IN THE MERCHANT BACK OFFICE

In the **Settings** > **Company** > **Merchant IDs** menu:

1. Click **Create a Merchant ID (MID)**.
   The "Selection of Merchant ID type" dialog box appears.



2. Select **Google Pay**.
   The Merchant ID creation assistant window appears.



   Gateway and **Gateway Merchant ID** are populated. These two pieces of information are useful for integrating Google Pay™ into a mobile application.

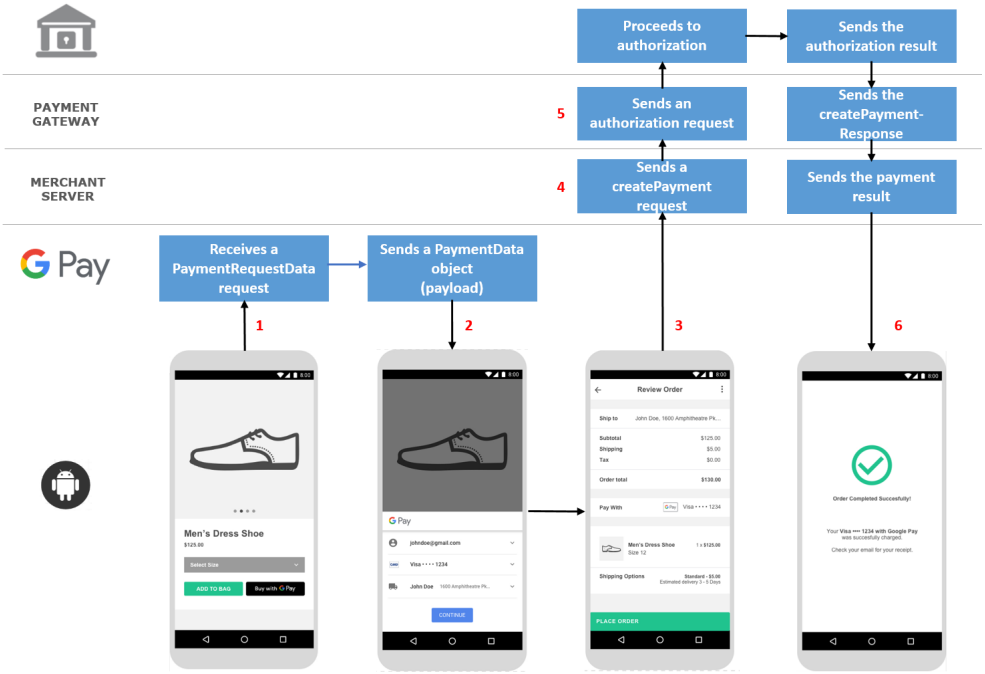
   Contact the PayZen if you would like to configure the CVV entry in the mobile integration.

3. Read and accept the Google Pay™ Terms of Service by ticking the box provided for this purpose.
   This action is mandatory for finalizing the creation of the MID.

4. Click **Terminate**.

Once the MID has been created, click "**Associate with a shop**" and associate it with the store of your choice.

6. EXCHANGE PROCESS



1. The mobile application submits a `paymentDataRequest` request to the Google Pay™ API to collect the buyer card data.
2. The Google Pay™ API returns the `PaymentData` object containing the encrypted data (also called "payload").
3. The mobile application transmits the payload to the merchant server.
4. The merchant server creates and submits a `createPayment` request using the payload to populate the **walletPayload** property of the `cardRequest` object. The `scheme` property is set to **GOOGLEPAY**.

   The other properties of the cardRequest object must not be populated.
5. The payment gateway analyzes the request, decrypts the card data and proceeds to the authorization request. It then transmits the payment result to the merchant server.
6. The merchant server receives a `createPaymentResponse` and analyzes the payment result. It transmits the information to the app, which will process it.

7. MANAGING THE TIMEOUT

Processing a web service query revolves around a series of asynchronous events such as:

- sending the query via the network of the merchant website,
- transmitting information on the Internet,
- payment processing by the payment gateway,
- soliciting bank servers, etc.

An incident may occur at any of these steps which would slow down the processing (and, therefore, the delay for the buyer).

The response to a query may be slowed down for several reasons:

- The response from the cardholder's card issuer is taking a long time (in case of foreign cards, periods of heavy workload, e.g. during sales, etc.).
- The response from the acquirer when transmitting and receiving the authorization request is taking a long time.
- The response from your application is taking a long time due to heavy workload.
- The response from the payment gateway is taking a long time.
- A peering problem on the Internet which may lead to losing messages, etc.

Depending on the configured timeouts, you may have to wait for an answer while the asynchronous processing is still performed by the payment gateway.

Long processing time should not be considered a declined payment.

For this reason, you must configure your code to handle potential problems that may occur when connecting to the SOAP API.

Advice

The average time for processing a payment request by the payment gateway is less than 5 seconds.

You must configure a timeout of 20 to 30 seconds between:

- the merchant server and the payment gateway,
- the mobile application and the merchant server.

During the waiting time, a message must clearly state to the buyer that their payment is in progress.

Once the timeout expires, you must indicate to the buyer that their payment has been rejected.

In order to correctly generate these timeout cases, we recommend 2 solutions:

- **Creating your transactions in manual validation mode**

  The merchant server must return a request to validate each successful transaction.

  The transactions made after the timeout will not be validated and will expire (after 7 days for a payment performed within the CB network).

  Once expired, they will not be able to be captured at the bank.

  The buyer will not be debited.

- **Creating your transactions in automatic validation mode**

The merchant server must interrogate the payment gateway to cancel the transaction or refund the buyer for each request received after the timeout.

For this, the merchant server will have to make sure to transmit a unique order identifier in the payment requests.

Then, for each transaction received after the timeout, the merchant server must

- interrogate the payment gateway to retrieve the UUID and the status of the corresponding transaction (`findPayments` method)

  If the UUID does not exist, it means that a technical error has made creating the transaction impossible.

  If the transaction has a REFUSED status, no action is necessary.

- cancel or refund the transaction (using the `cancelPayment` or `refundPayment` methods) if the authorization request was accepted during the timeout.

8. VIEWING SAMPLE CODES

To provide you with support for each application, we have provided additional examples of implementation. Please visit the links below:

Merchant server

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

Android

- *https://github.com/lyra/googlepay-payment-android-integration-sample*

9. CONFIGURING YOUR PROJECT

For more information on the Google Pay™ payment API, see the official documentation: *https://developers.google.com/pay/api/android/*

Follow Google recommendations for visual elements (logos and texts):

https://developers.google.com/pay/api/android/guides/brand-guidelines

If you encounter problems with calls to Google Pay™ API, go to the following page:

https://developers.google.com/pay/api/android/support/troubleshooting

9.1. Activating the API Google Pay™

To enable the Google Pay™ API and use it in your project, you have to add the following to the **<application>** tag of the AndroidManifest.xml file:

```
<application>
 ...
 <!-- Enables the Google Pay API -->
 <meta-data
  android:name="com.google.android.gms.wallet.api.enabled"
  android:value="true" />
</application>
```

9.2. Adding dependencies

Add the Google Play Services 16.0.x (minimum version required) to your build.gradle file:

```
dependencies {
 compile 'com.google.android.gms:play-services-wallet:16.0.0'
 compile 'com.android.support:appcompat-v7:24.1.1'
}
```


10. PROCEEDING TO PAYMENT VIA THE MOBILE APPLICATION

The steps below show how to integrate the Google Pay™ API in a mobile application.

10.1. Defining the version of Google Pay™

Create a **BaseRequest** object containing the properties below that will be used in all your queries:

Excerpt from the code sample:

```
private fun getBaseRequest(): JSONObject {
 return JSONObject()
  .put("apiVersion", 2)
  .put("apiVersionMinor", 0)
}
```

10.2. Defining the payment gateway

Implement the **getTokenizationSpecification** method as shown below in order to specify that you are using the "lyra" payment gateway.

Excerpt from the code sample:

```
private const val GATEWAY_TOKENIZATION_NAME = "lyra"
private const val GATEWAY_MERCHANT_ID = "<REPLACE_ME>"

private fun getTokenizationSpecification(gatewayMerchantId: String): JSONObject {
 val params = JSONObject().put("gateway", GATEWAY_TOKENIZATION_NAME)
 params.put("gatewayMerchantId", gatewayMerchantId)

 return JSONObject().put("type", "PAYMENT_GATEWAY").put("parameters", params)
}
```

*Replace <REPLACE_ME> by the Google Pay™ contract number that can be found in (**Settings** > **Company** > **Merchant IDs**).*

If you are using the sample code, enter your Google Pay™ contract number in the MainActivity.kt file (`GATEWAY_MERCHANT_ID` variable).

10.3. Defining supported payment methods

Implement the **getAllowedCardNetworks** method as shown below in order to specify the card types accepted for payments.

Excerpt from the code sample:

```
private const val SUPPORTED_NETWORKS = "AMEX, VISA, MASTERCARD, DISCOVER, JCB"
this.supportedNetworks = supportedNetworks.split(Regex(",[ ]*")).toTypedArray()

private fun getAllowedCardNetworks(): JSONArray {
    val allowedCardNetworks = JSONArray()
    for (network in supportedNetworks) {
        allowedCardNetworks.put(network)
    }
    return allowedCardNetworks
}
```

Implement the **`getAllowedCardAuthMethods`** method and assign the **PAN_ONLY** value to it.

Excerpt from the code sample:

```
private val SUPPORTED_METHODS = Arrays.asList("PAN_ONLY")!!

private fun getAllowedCardAuthMethods(): JSONArray {
    val allowedCardAuthMethods = JSONArray()
    for (method in SUPPORTED_METHODS) {
        allowedCardAuthMethods.put(method)
    }
    return allowedCardAuthMethods
}
```

For more information, see the documentation of the CardParameters object.

10.4. Transmitting supported payment methods

Implement the **`getCardPaymentMethod`** function.

Create a `cardPaymentMethod` object to transmit the supported payment methods for the `CARD` payment method as well as the payment gateway details.

Excerpt from the code sample:

```
private fun getCardPaymentMethod(additionalParams: JSONObject?, tokenizationSpecification:
 JSONObject?): JSONObject {
    val params = JSONObject()
            .put("allowedAuthMethods", getAllowedCardAuthMethods())
            .put("allowedCardNetworks", getAllowedCardNetworks())

    // Additional parameters provided?
   if (additionalParams != null && additionalParams.length() > 0) {
        val keys = additionalParams.keys()
        while (keys.hasNext()) {
            val key = keys.next()
            params.put(key, additionalParams.get(key))
        }
    }
    val cardPaymentMethod = JSONObject()
    cardPaymentMethod.put("type", "CARD")
    cardPaymentMethod.put(
            "parameters", params)
    if (tokenizationSpecification != null) {
        cardPaymentMethod.put(
                "tokenizationSpecification", tokenizationSpecification)
    }
    return
}
```

For more information, see the documentation of the CardParameters object.

10.5. Creating a PaymentsClient instance

Create an instance of `PaymentsClient` in the `onCreate` method of your `Activity` to allow interaction with the `PaymentsClient` API.

```
private fun initPaymentsClient(activity: Activity, mode: String): PaymentsClient {
 val builder = Wallet.WalletOptions.Builder()
 if (mode == "TEST") {
  builder.setEnvironment(WalletConstants.ENVIRONMENT_TEST)
 } else {
  builder.setEnvironment(WalletConstants.ENVIRONMENT_PRODUCTION)
 }
 return Wallet.getPaymentsClient(activity, builder.build())
}
```


During the integration phase, if you are using the sample code, the `PAYMENT_MODE` variable must be set to **TEST** in the MainActivity.kt file.

10.6. Verifying if payment is possible

Implement the **isReadyToPayRequest** method to check if the Android and Google Play versions installed on the mobile device are supported by the PaymentsClient API.

Depending on the response, it will be possible to hide/show the payment button.

Excerpt from the code sample:

```
private fun prepareIsReadyToPayRequest(): IsReadyToPayRequest {
 val isReadyToPayRequest = getBaseRequest()
 isReadyToPayRequest.put(
  "allowedPaymentMethods", JSONArray()
  .put(getCardPaymentMethod(null, null)))
 return IsReadyToPayRequest.fromJson(isReadyToPayRequest.toString())
}

PayZenPayment.isPaymentPossible(paymentsClient).addOnCompleteListener { task ->
 try {
  val result = task.getResult(ApiException::class.java)
  if (result) {
   // show Google Pay as a payment option
   payBtn.visibility = View.VISIBLE
  } else {
   payBtn.visibility = View.VISIBLE
   Toast.makeText(this, "isPaymentPossible return false", Toast.LENGTH_LONG).show()
  }
 } catch (e: ApiException) {
  Toast.makeText(this, "isPaymentPossible exception catched", Toast.LENGTH_LONG).show()
 }
}
```

10.7. Creating a PaymentDataRequest object

When the buyer is ready to pay, create a `PaymentDataRequest` object and assign the **BaseRequest** object to it.

This object should contain information that will be used for the payment:

- payment gateway details,
- payment methods supported by your application,
- additional information that you wish to obtain in the response.

Define the supported payment methods by populating the `allowedPaymentMethods` property with the result of the `getCardPaymentMethod` function (see chapter Transmitting supported payment methods on page 14).

Define transaction details in the `TransactionInfo` object:

property	totalPriceStatus
value	**"FINAL"**
	Required parameter when making the `createPaymentDataRequest()` call. Its value will not be taken into account for the payment.
property	**totalPrice**
value	"10.00"
	Required parameter when making the `createPaymentDataRequest()` call. However, only the value transmitted during the call to the `createPayment` method of the SOAP Web service API v5 will be taken into account for the payment.
property	**currencyCode**

value	"EUR"
	Required parameter when making the `createPaymentDataRequest()` call. However, only the value transmitted during the call to the createPayment method of the SOAP Web service API v5 will be taken into account for the payment.

For more information, see the documentation of the TransactionInfo object.

Specify the additional information that you wish to obtain in the response:

property	billingAddressRequired
value	true
	Makes the entry of the billing address mandatory. The selected address will be returned in the payload. Useful if you wish to offer express checkout to the buyer.
property	**emailRequired**
value	true
	Allows to transmit the e-mail associated with the selected Google account.
property	**shippingAddressRequired**
value	true
	Makes the entry of the shipping address mandatory. The address selected by the buyer will be returned in the payload. Useful if you wish to offer express checkout to the buyer.
property	**phoneNumberRequired**
value	true
	Makes the entry of the phone number mandatory within the billing and shipping address. If `shippingAddressRequired` or `billingAddressRequired` is set to `true`, the phone number will be transmitted in the payload. Useful if you wish to offer express checkout to the buyer.

For more information, see the documentation of the PaymentDataRequest object.

Excerpt from the sample code:

```
private fun preparePaymentDataRequest(price: String, currency: String, gatewayMerchantId:
 String): PaymentDataRequest {
    val paymentDataRequestJson = getBaseRequest()
    val additionalParams = JSONObject()
    val transactionJson = JSONObject()
    transactionJson
            .put("totalPriceStatus", "FINAL")
            .put("totalPrice", price)
            .put("currencyCode", currency)
    additionalParams.put("billingAddressRequired", true)
    additionalParams
            .put("billingAddressParameters", JSONObject()
            .put("format", "FULL").put("phoneNumberRequired", false))
    paymentDataRequestJson
            .put("allowedPaymentMethods", JSONArray()
            .put(getCardPaymentMethod(additionalParams,
 getTokenizationSpecification(gatewayMerchantId))))

    paymentDataRequestJson.put("shippingAddressRequired", true)
    paymentDataRequestJson.put("emailRequired", true)
    paymentDataRequestJson.put("transactionInfo", transactionJson)
    return PaymentDataRequest.fromJson(paymentDataRequestJson.toString())
}
```


10.8. Registering an event handler for user gesture

Define a `OnClickListener` for your "Pay" button in order to trigger the creation of a `PaymentDataRequest` object. Use the `resolveTask` method of the `AutoResolveHelper` with the `PaymentDataRequest` in order to show the Google Pay Bottom Sheet.

The result will then be transmitted to your onActivityResult() method.

Excerpt from the sample code:

```
fun onPayClick(view: View) {
 progressBar.visibility = View.VISIBLE

 val paymentDataRequest = preparePaymentDataRequest()
 AutoResolveHelper.resolveTask(
  paymentsClient.loadPaymentData(paymentDataRequest),
  this,
  GooglePayManagement.GOOGLE_PAYMENT_CODE_RESULT
 )
}
```

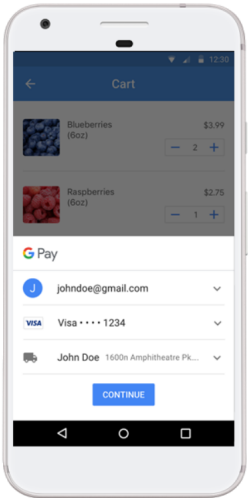

10.9. Selecting a payment method

In order to be able to pay with Google Pay™, the buyer must have recorded a valid payment method in his Google Pay™ wallet.

To complete the integration phase in test environment, the integrator must:

- have a Google account,
- associate a valid payment method with their Google Pay™ account.

When the Google Pay™ bottom sheet is displayed, select your card and click the Continue button.



In test environment, the data returned in the PaymentData object is fictional and will not match the data of the card registered on your Google account.

Your card will never be debited as long as your application is configured in TEST environment.

In test environment, the following message will appear on the Google Pay bottom sheet:
"*Unrecognized app. Please make sure that you trust this app before proceeding.*"

10.10. Retrieving data from Google Pay™

The buyer data is returned to the mobile application in a `PaymentData` object (also called “payload”).

In order to process the payload data, override the `onActivityResult`() method of your `activity` as follows:

Excerpt from the code sample:

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    // Manage Google Pay result
   if (requestCode == GooglePayManagement.GOOGLE_PAYMENT_CODE_RESULT) {
        when (resultCode) {
            Activity.RESULT_OK -> {
                if (data != null) {
                    val paymentData = PaymentData.getFromIntent(data)
                    val googlePayData = paymentData!!.toJson()
                    // Execute payment
                 PayZenPayment.executeTransaction(googlePayData, this)
                } else {
                    PayZenPayment.returnsResult(false,
 PayZenPaymentErrorCode.UNKNOWN_ERROR, "Unknown error", this)
                }
            }
            Activity.RESULT_CANCELED -> {
```



```
                PayZenPayment.returnsResult(false,
 PayZenPaymentErrorCode.PAYMENT_CANCELLED_ERROR, "Payment cancelled by user", this)
            }
            AutoResolveHelper.RESULT_ERROR -> {
                PayZenPayment.returnsResult(false,
 PayZenPaymentErrorCode.UNKNOWN_ERROR, "Unknown error", this)
            }
        }
    }
}
```

If you have opted for express checkout, you must transform the payload into a JSON object and analyze it in order to save the necessary data (buyer contact details, shipping address, etc.)

Example of retrieving the shipping address:

```
when (resultCode) {
 Activity.RESULT_OK -> {
  if (data != null) {
   val paymentData = PaymentData.getFromIntent(data)
   val googlePayData = paymentData!!.toJson()
   val billingAddress =
       JSONObject(
        JSONObject(
         JSONObject(
          googlePayload!!.toJson()
         ).getString("paymentMethodData")
        ).getString("info")
       ).getString("billingAddress")
```

Example of a PaymentData object:

```
JSON result Example :{
 "apiVersionMinor":0,
 "apiVersion":2,
 "paymentMethodData":{
  "description":"Visa •••• 3513",
  "tokenizationData":{
   "type":"PAYMENT_GATEWAY",
   "token":"{\"signature\":\"MEUCIC1+XSqhtgCWQa\/3k7OFIesJX0SYBtCeOKjD
+O2JBWO9AiEAg03FhXVVwD0NRhiPnjGWdfTqTXm39fnzr+XvHdbecY4\\u003d\",\"protocolVersion
\":\"ECv1\",\"signedMessage\":\"{\\\"encryptedMessage\\\":\\\"qjXoUw7ptqWE\/G
+PSOb8QoR+hqzM7h1S7iQ3jH\/mxmjJi\/\/MY5Kem6WmuQxMQbz0pXeX5mxjmUxvbIQzGhtLLELoTWxuAF
\/mLSuoIznPbvJNxQ9xGOVMQYuo0jTKqUUv6QrjMBTSCcQx2ktkDS3EjHOWBPTw22mI\/
kXz0wn5x6jlMLEUTFkNlPZYjeoFRvUXpw0XoECNdouiiJwub52zmyebY7pz1JlG5DQOMWRvrCnj5Kgs9sxzXp1VJJ
+rIqewoBOgl\/BgKsf87fOxVJFYqOmq+LR\/wk8PDKE9VE1XakQBG8kxhSflMQaqC
+ObObJfoUC9ruIDK7iFkT8EFV3FVWHu8ZVYmoek3HWabZ36NdRPUyGtDCjOR+CmT8tneRHyqRzy8S4Yk5LyCObTZ
+rbI0toveNchZdakUhMLyFFLqkgnc\/g3WJnLpujRy3jgOe6\/+ymb1gaiOnQQA\\\\u003d\\\\u003d\\
\",\\\"ephemeralPublicKey\\\":\\\"BKB3qFuCrmNI0bOxVg\/kVCcCtswHrwabcrm18JtPTB+w5L\/
d39dnrgaGajdYvdLrFRQc5RyQ52Ug3e151yDNE6E\\\\u003d\\\",\\\"tag\\\":\\\"rJ5SapWOGc\/f4eqzif+P9D\/
wSg\/yvfOJ63yVbRc1lGY\\\\u003d\\\"}\"}"
  },
  "type":"CARD",
  "info":{
   "cardNetwork":"VISA",
   "cardDetails":"3513",
   "billingAddress":{
    "address3":"",
    "sortingCode":"",
    "address2":"",
    "countryCode":"FR",
    "address1":"109 Rue de l'Innovation",
    "postalCode":"31670",
    "name":"My Network",
    "locality":"Labège",
    "administrativeArea":""
   }
  }
 },
 "shippingAddress":{
  "address3":"",
  "sortingCode":"",
  "address2":"",
  "countryCode":"FR",
  "address1":"109 Rue de l'Innovation",
  "postalCode":"31670",
  "name":"My Network",
  "locality":"Labège",
```



```
  "administrativeArea":""
 },
 "email":"network.gpay@gmail.com"
}
```


11. TRANSMITTING DATA TO THE PAYMENT GATEWAY

After the mobile application retrieves the payment data, you must transmit it to your server.

Once the data is received on the server side, you must call the payment creation Web Service (`createPayment`) of the SOAP Web service API v5

Your query must consist of the following elements:

SOAP headers	Required	Allows to identify the merchant and to secure the exchanges with the payment gateway.
A **commonRequest** object	Optional	Allows to force the contract number to be used for the payment.
A **cardRequest** object	Required	Allows to transmit the payload to the payment gateway. Once decrypted, the data will be used to perform the payment.
A **paymentRequest** object	Required	Allows to transmit the payment details (amount, currency, etc.)
A **customerRequest** object	Optional	Allows to transmit the buyer details
A **billingRequest** sub-object	Optional	Allows to transmit the buyer's billing address
A **shippingRequest** sub-object	Optional	Allows to transmit the buyer's shipping address
A **shoppingCartRequest** object	Optional	Allows to transmit information about the content of the shopping cart

Note:

The current Google Pay™ implementation is not compatible with 3D Secure. Therefore, there is no need to transmit the **threeDSRequest** object in your query.

Step 1: Generate the `SOAP-HEADER` as defined in the documentation.

Example of a header:

```
<?xml version="1.0" encoding="UTF-8"?>
  <SOAP-ENV:Enveloppe xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1="http://v5.ws.vads.lyra.com/"
    xmlns:ns2="http://v5.ws.vads.lyra.com/Header/">
  <SOAP-ENV:Header>
    <ns2:shopId>12345678</ns2:shopId>
    <ns2:requestId>70c08d51-214b-4a9b-9e74-eadcdde6710c</ns2:requestId>
    <ns2:timestamp>2018-07-06T13:51:32Z</ns2:timestamp>
    <ns2:mode>TEST</ns2:mode>
    <ns2:authToken>o53KmUXgjqhiyTooIfRjh6wF+NJwVQg790mpbrjTg18=</ns2:authToken>
  </SOAP-ENV:Header>
...
```

Step 2: Create a **commonRequest** object as defined in the documentation if you wish to force the number of the MID to use for the payment.

Step 3: Create a **cardRequest** object:

Attributes of the cardRequest object	Value
scheme	**GOOGLEPAY**
walletPayload	The `PaymentData` object in json format as it was received by the merchant server.

The **number**, **expiryYear** and **expiryMonth** attributes become optional when the **scheme** attribute is populated with **GOOGLEPAY**.

Step 4: Create a **paymentRequest** object:

Attributes of the cardRequest object	Description
amount	Amount to be paid. May be different from the value transmitted in the `PaymentDataRequest` object.

Attributes of the cardRequest object	Description
currency	Numeric code of the currency to be used for the payment. E.g.: E.g.: 978 for euro (EUR)

Step 5: Create a **orderRequest** object:

Attribute of the orderRequest object	Description
orderId	Order reference. It is recommended to use unique order references. In case of timeout, this reference will be used to find the corresponding transaction.

Step 6: **Manage shipping and billing addresses**.

If you have opted for express checkout, the data will be automatically transmitted to the payment gateway. You can move on to step 7.

If you manage address selection via your mobile application, you can transmit this information to the payment gateway so that it becomes visible in the Back Office. To do this, create a `customerRequest` object.

Step 6.1: Create a **billingRequest** object:

Attributes of the billingRequest object	Description
email	Buyer's e-mail address. If you leave this field empty, the payment gateway will automatically retrieve the "**email**" property value of the payload.
address	First line of the billing address. If you leave this field empty, the payment gateway will automatically retrieve the "**paymentMethodData.Info.billingAddress.address1**" property value of the payload.
zipCode	Zip code of the billing address. If you leave this field empty, the payment gateway will automatically retrieve the "**paymentMethodData.Info.billingAddress.postalCode**" property value of the payload.
city	City of the billing address. If you leave this field empty, the payment gateway will automatically retrieve the "**paymentMethodData.Info.billingAddress.locality**" property value of the payload.
country	Country of the billing address. If you leave this field empty, the payment gateway will automatically retrieve the "**paymentMethodData.Info.billingAddress.countryCode**" property value of the payload.
firstName	Name of the billing address. If you leave this field empty, the payment gateway will automatically retrieve the "**paymentMethodData.Info.billingAddress.name**" property value of the payload.
cellPhoneNumber	Cell phone number If you leave this field empty, the payment gateway will automatically retrieve the "**paymentMethodData.Info.billingAddress.phoneNumber**" property value of the payload.

Step 6.2: Create a **shippingRequest** object:

Attributes of the shippingRequest object	Description
email	Buyer's e-mail address. If you leave this field empty, the payment gateway will automatically retrieve the "**email**" property value of the payload.
address	First line of the billing address. If you leave this field empty, the payment gateway will automatically retrieve the "**shippingAddress.address1**" property value of the payload.
zipCode	Zip code of the billing address. If you leave this field empty, the payment gateway will automatically retrieve the "**shippingAddress.postalCode**" property value of the payload.
city	City of the billing address. If you leave this field empty, the payment gateway will automatically retrieve the "**shippingAddress.locality**" property value of the payload.
country	Country of the billing address. If you leave this field empty, the payment gateway will automatically retrieve the "**shippingAddress.countryCode**" property value of the payload.
firstName	Name of the billing address. If you leave this field empty, the payment gateway will automatically retrieve the "**shippingAddress.name**" property value of the payload.
cellPhoneNumber	Cell phone number If you leave this field empty, the payment gateway will automatically retrieve the "**shippingAddress.phoneNumber**" property value of the payload.

Step 7: Create a **shoppingCartRequest** object if you wish to transmit the information on the shopping cart content.

See Implementation guide - SOAP Web service API v5 for more details on the integration of the createPayment() method.

Example of an XML message generated during the call to the createPayment() method:

```
<?xml version="1.0" encoding="UTF-8"?>
  <SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1="http://v5.ws.vads.lyra.com/"
    xmlns:ns2="http://v5.ws.vads.lyra.com/Header/">
  <SOAP-ENV:Header>
    <ns2:shopId>12345678</ns2:shopId>
    <ns2:requestId>88c53eb7-6015-4ec6-8045-206e5caa2e2c</ns2:requestId>
    <ns2:timestamp>2018-07-09T10:15:54Z</ns2:timestamp>
    <ns2:mode>TEST</ns2:mode>
    <ns2:authToken>ALxESNJ+mE5htqWccF5OTkYC2noxnDr570DrnI2+5SE=</ns2:authToken>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <ns1:createPayment>
      <commonRequest>
        <paymentSource>EC</paymentSource>
      </commonRequest>
      <paymentRequest>
        <amount>2990</amount>
        <currency>978</currency>
      </paymentRequest>
      <orderRequest>
        <orderId>myOder</orderId>
      </orderRequest>
      <cardRequest>
        <scheme>GOOGLEPAY</scheme>
        <walletPayload>{"apiVersionMinor":0,"apiVersion":2,"paymentMethodData":{"description":
        "Visa •••• 3513","tokenizationData":{"type":"PAYMENT_GATEWAY","token":"{\"signature\":
        \"MEUCIC1+XSqhtgCWQa\/3k7OFIesJX0SYBtCeOKjD+O2JBWO9AiEAg03FhXVVwD0
        NRhiPnjGWdfTqTXm39fnzr+XvHdbecY4\\u003d\",\"protocolVersion\":\"ECv1\",\"signedMessage
\":\"
        {\\\"encryptedMessage\\\":\\\"qjXoUw7ptqWE\/G+PSOb8QoR+hqzM7h1S7iQ3jH\/mxmjJi\/\/M
        Y5Kem6WmuQxMQbz0pXeX5mxjmUxvbIQzGhtLLELoTWxuAF\/mLSuoIznPbvJNxQ9xGOVM
      QYuo0jTKqUUv6QrjMBTSCcQx2ktkDS3EjHOWBPTw22mI\/kXz0wn5x6jlMLEUTFkNlPZYjeoFR
      vUXpw0XoECNdouiiJwub52zmyebY7pz1JlG5DQOMWRvrCnj5Kgs9sxzXp1VJJ+rIqewoBOgl\/B
      gKsf87fOxVJFYqOmq+LR\/wk8PDKE9VE1XakQBG8kxhSflMQaqC+ObObJfoUC9ruIDK7iFkT8
      EFV3FVWHu8ZVYmoek3HWabZ36NdRPUyGtDCjOR+CmT8tneRHyqRzy8S4Yk5LyCObTZ+rb
      I0toveNchZdakUhMLyFFLqkgnc\/g3WJnLpujRy3jgOe6\/+ymb1gaiOnQQA\\\\u003d\\\\u003d\\\",\\\"e
      phemeralPublicKey\\\":\\\"BKB3qFuCrmNI0bOxVg\/kVCcCtswHrwabcrm18JtPTB+w5L\/d39dnrga
      GajdYvdLrFRQc5RyQ52Ug3e151yDNE6E\\\\u003d\\\",\\\"tag\\\":\\\"rJ5SapWOGc\/f4eqzif+P9D\/wS
      g\/yvfOJ63yVbRc1lGY\\\\u003d\\\"}\"}"},"type":"CARD","info":
{"cardNetwork":"VISA","cardDetails":
      "3513","billingAddress":
{"address3":"","sortingCode":"","address2":"","countryCode":"FR","address1"
      :"109Ruedel'Innovation","postalCode":"31670","name":"Lyranetwork","locality":"Labège",
      "administrativeArea":""}}},"shippingAddress":
{"address3":"","sortingCode":"","address2":"",

 "countryCode":"FR","address1":"109Ruedel'Innovation","postalCode":"31670","name":"MyNetwork",
      "locality":"Labège","administrativeArea":""},"email":"test.gpay@gmail.com"}</
walletPayload>
      </cardRequest>
      <customerRequest>
        <billingDetails>
          <email>test@test.com</email>
        </billingDetails>
        </customerRequest>
        <techRequest/>
      </ns1:createPayment>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


12. PROCESSING THE PAYMENT RESULT

When the payment gateway receives your createPayment request, it performs different verification processes.

These processes may result in:

- exceptions (SOAP Fault Exception)
- application errors

In case of an exception, the response will contain a `Fault` object specifying the encountered error (format error, etc.).

Example of a response when the shop does not have the options required for Google Pay payment:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"/>
   <soap:Body>
      <soap:Fault>
         <faultcode xmlns:ns1="http://www.w3.org/2003/05/soap-envelope">ns1:Sender</faultcode>
         <faultstring>
              bad.shopId: The shop with shopId 12345678 is not allowed to call the Web Service
         </faultstring>
         <detail>
            <requestId>f269ff49-3b8a-4314-b999-24b39ce03287</requestId>
         </detail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

If the request is valid, the response contains the `commonReponse` object.

The `commonReponse.responseCode` field gives an indication on request processing.

When `responseCode` is set to 0 in the response, the transaction has been created.

Any values other than `responseCode` indicate that the request has been rejected before the payment.

The `responseCodeDetail` field provides the details of the encountered error.

To verify the payment status, you must analyze the value of the `commonResponse.`**`transactionStatusLabel`** field

The only statuses that can guarantee an accepted payment are:

- **CAPTURED**
- **AUTHORIZED**
- **AUTHORIZED_TO_VALIDATE**

When `transactionStatusLabel` is set to `REFUSED`, you must verify the presence of the `paymentError` field in the `paymentResponse` object.

This field provides indications on the reason of the refusal.

If `paymentError` is not populated, analyze the `result` field of the `authorizationResponse` object to know the reason of payment rejection.

See Implementation guide - SOAP Web service API v5 to obtain the complete list of possible errors.

Example of a response returned in case of accepted payment

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns1="http://v5.ws.vads.lyra.com/"
   xmlns:ns2="http://v5.ws.vads.lyra.com/Header/" ">
<SOAP-ENV:Header>
   <shopId>12345678</shopId>
   <requestId>8bb621fa-8512-45cc-b695-a30fe104cb06</requestId>
   <timestamp>2018-07-11T09:51:00Z</timestamp>
   <mode>TEST</mode>
   <authToken>Le5sPmTnoE8xh3qtU7F1SxhaieO3rh9uLxQpy/4bEu4=</authToken>
</SOAP-ENV:Header>
<soap:Body>
   <ns2:createPaymentResponse>
      <createPaymentResult>
         <requestId>8bb621fa-8512-45cc-b695-a30fe104cb06</requestId>
         <commonResponse>
            <responseCode>0</responseCode>
            <responseCodeDetail>Action successfully completed</responseCodeDetail>
            <transactionStatusLabel>AUTHORISED</transactionStatusLabel>
            <shopId>91335531</shopId>
            <paymentSource>EC</paymentSource>
            <submissionDate>2018-07-11T11:51:00.749+02:00</submissionDate>
            <contractNumber>5555555</contractNumber>
         </commonResponse>
         <paymentResponse>
            <transactionId>927452</transactionId>
            <amount>2990</amount>
            <currency>978</currency>
            <expectedCaptureDate>2018-07-11T11:51:00.791+02:00</expectedCaptureDate>
            <operationType>0</operationType>
            <creationDate>2018-07-11T11:51:00.791+02:00</creationDate>
            <liabilityShift>NO</liabilityShift>
            <transactionUuid>3257eebe44ab4b409315022735c45c2a</transactionUuid>
            <sequenceNumber>1</sequenceNumber>
            <paymentType>SINGLE</paymentType>
         </paymentResponse>
         <orderResponse>
            <orderId>myOder</orderId>
         </orderResponse>
         <cardResponse>
            <number>411111XXXXXX1111</number>
            <scheme>VISA</scheme>
            <brand>VISA</brand>
            <country>GB</country>
            <bankCode>0169</bankCode>
            <expiryMonth>12</expiryMonth>
            <expiryYear>2023</expiryYear>
         </cardResponse>
         <authorizationResponse>
            <mode>FULL</mode>
            <amount>2990</amount>
            <currency>9788</currency>
            <date>2018-07-11T11:51:00.791+02:00</date>
            <number>3fe3c2</number>
            <result>0</result>
         </authorizationResponse>
         <captureResponse/>
         <customerResponse>
            <billingDetails>
               <language>fr_FR</language>
            </billingDetails>
            <shippingDetails/>
            <extraDetails/>
         </customerResponse>
         <markResponse/>
         <threeDSResponse>
            <authenticationResultData>
            <transactionCondition>COND_SSL</transactionCondition>
            </authenticationResultData>
         </threeDSResponse>
         <extraResponse/>
         <fraudManagementResponse>
            <riskControl>
            <name>CARD_FRAUD</name>
            <result>OK</result>
         </riskControl>
         </fraudManagementResponse>
      </createPaymentResult>
   </ns2:createPaymentResponse>
</soap:Body>
</soap:Envelope>
```


13. SHIFTING TO PRODUCTION MODE

If your integration is correctly completed, you will successfully create a Google Pay transaction in test environment.

You must now get your development validated via Google.

1. Google provides a checklist:
   https://developers.google.com/pay/api/android/guides/test-and-deploy/integration-checklist
2. Once all the points in the checklist are validated, you must fill the following form to request the shift into production mode:
   https://services.google.com/fb/forms/googlepayAPIenable/
3. When Google asks you for your production application, modify the `onCreate` method of your `Activity` as follows:

```
...
      builder.setEnvironment(WalletConstants.ENVIRONMENT_PRODUCTION)
...
```

   If you are using the sample code, change the value of the `PAYMENT_MODE` variable to **PRODUCTION** in the MainActivity.kt file.

4. Then, modify the `createPayment` call made by your merchant server in order to use the production key in the `SOAP HEADER` generation.
5. Modify the value of the `mode` header in your `createPayment` method as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
  <SOAP-ENV:Enveloppe xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1="http://v5.ws.vads.lyra.com/"
    xmlns:ns2="http://v5.ws.vads.lyra.com/Header/">
  <SOAP-ENV:Header>
    <ns2:shopId>70258842</ns2:shopId>
    <ns2:requestId>b2e0beab-371e-4751-9aa7-20d69daac1ac</ns2:requestId>
    <ns2:timestamp>2018-07-09T12:37:51Z</ns2:timestamp>
    <ns2:mode>PRODUCTION</ns2:mode>
    <ns2:authToken>x7JqR7QLDRc4bsM57nOyf5xzKqlalEqPmeai1EAOZDM=</ns2:authToken>
  </SOAP-ENV:Header>
```

6. Send your application configured in production mode to Google to complete the last tests.
   The application must be signed with the release key. The debug key will not work in the production environment.
7. After receiving an authorization from Google, enable Google Pay payments in the Google Pay Developer Profile for this application, then deploy the application in the Google Play Store.
   For more details, see the application deployment process:
   https://developers.google.com/pay/api/android/guides/test-and-deploy/deploy-your-application?authuser=1

After shifting into production mode, if you make a call with the **WalletConstants.ENVIRONMENT_TEST**, the following message will appear on the Google Pay bottom sheet:
"*Unrecognized app. Please make sure that you trust this app before proceeding.*"

14. MANAGING YOUR GOOGLE PAY™ TRANSACTIONS VIA THE BACK OFFICE PAYZEN

The Back Office allows to perform different actions with Google Pay™ transactions depending on their status:

Actions available via the **Transactions in progress** tab:

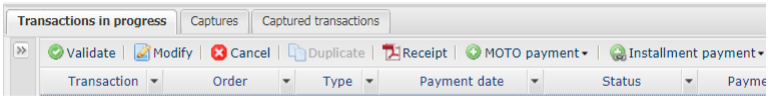
- Display transaction details
- Validate
- Modify
- Cancel
- Capture manually (only for transactions performed in test environment)
- Edit the order reference
- Resend transaction confirmation e-mail to the buyer
- Resend transaction confirmation e-mail to the merchant.

Actions available via the **Captured transactions** tab:

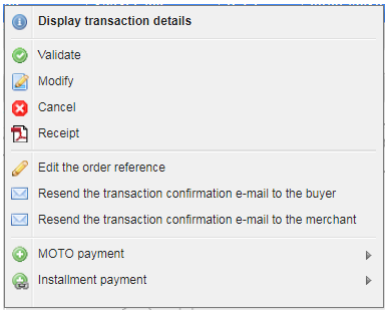
- Refund
- Duplicate
- Edit the order reference
- Resend transaction confirmation e-mail to the buyer
- Resend transaction confirmation e-mail to the merchant
- Manual reconciliation.

You can access these actions in three ways:

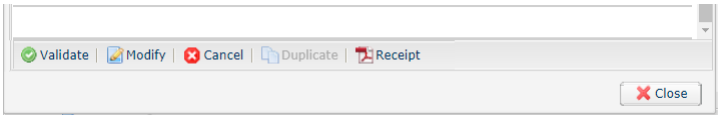
- via the menu bar



- using right-click



- at the bottom of the **Transaction details**.



14.1. Display transaction details Google Pay™

Transactions can be viewed in the Merchant Back Office via the **Management** > **Transactions** menu.

Via the **Management** menu, the merchant has access to real and TEST transactions.

Note:

Depending on the access rights, TEST transactions (example: developer profile) and/or real transactions (example: accountant profile) can be viewed.

The content of the **Transactions in progress** tab is displayed by default. All the transactions of the day are listed.

Characteristics of a payment made with Google Pay™:

Successful payments can be viewed in the Merchant Back Office, **Transactions in progress tab**.

Failed payments can be viewed in the Merchant Back Office, **Transactions in progress tab**.

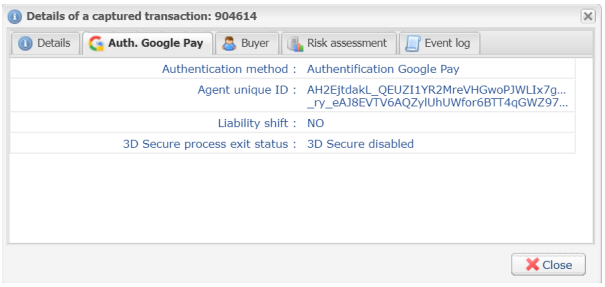
To view the details of a transaction:

1. Select a transaction.
2. Right click on it and select **Display transaction details** or double-click the transaction you wish to see the details of.

   The **Details of a transaction in progress** dialog box appears.

   The details include for example:

   - the payment method
   - the order reference
   - the transaction amount
   - the creation date of the transaction
   - the transaction status
   - The type of the digital wallet: Google Pay™
3. Click the **Google Pay Auth.** tab to view the Google Pay™ authentication details.



14.2. Validate a transaction

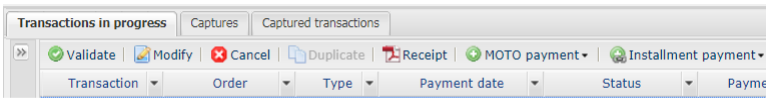
The **Validate** action is available when the transaction has one of the following statuses:

- To be validated
- To be validated and authorized

This action allows to validate the capture at the bank where the transaction took place. A transaction that has not been validated before the expiry date of the authorization request will get the expired status. It will not be captured at the bank.

In order to validate a transaction:

1. Display the tab **Transactions in progress**



2. Select the transaction.
3. Click on **Validate**

Once the transaction is validated, its status changes to:

- "**Waiting for capture**" for transactions with the "**To be validated**" status,
- "**Waiting for authorization**" for transctions with the "**To be validated and authorized**" status.

14.3. Modify a transaction

The **Modify** action is available when the transaction has one of the following statuses:

- To be validated
- To be validated and authorized
- Waiting for authorization
- Waiting for capture

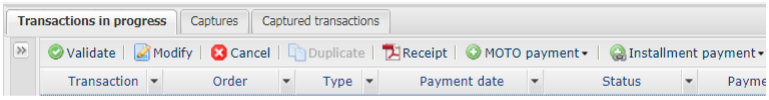
This action allows to modify the amount and the capture date at the bank with respect to the following constraints:

- the modified amount cannot be greater than the initial amount
- when the transaction has not yet been authorized, the capture date can be defined anytime between the current date and the capture date specified by the merchant during the payment.

  An authorization request will be automatically triggered if the selected capture date is between the current date and the expiry date of the authorization request (e.g.: period of 7 days for CB.
- when the transaction has already been authorized, the capture date at the bank cannot be later than the expiry date of the authorization (e.g.: 7 days for CB).

To modify a transaction:

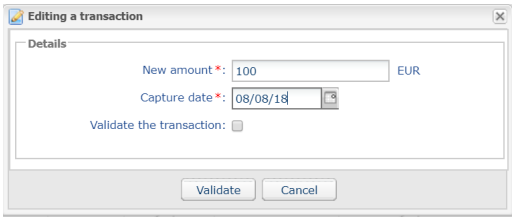
1. Display the tab **Transactions in progress**



2. Select the transaction.

3. Click **Modify**.

   The **Edition of a transaction** dialog box appears.



4. If you wish to modify the transaction amount, enter the new amount.

   Reminder: the new amount must be lower than the initial amount.

5. If you wish to modify the capture date at the bank, enter the capture date.

   The calendar will show the authorized slot for the capture date. The slot is calculated based on when the authorization expires. The authorization validity period depends on the payment method and the network that was used for the authorization request (e.g.: 7 days for CB).

   It is also possible to validate a transaction with the **To be validated** or **To be validated and authorized** status, by checking **Validate the transaction**.

6. Click **Validate**.

If you wish, you may view the transaction details to see the applied changes (right-click the edited transaction **Display transaction details**).

14.4. Cancel a transaction

The **Cancel** option is only available for the transactions that have not been captured.

1. Select a transaction with a right-click.
2. Select **Cancel**.
3. Confirm that you wish to definitively cancel the selected transaction.

The transaction status changes to **Canceled**.

Note

*It is possible to **cancel** several transactions at the same time.*

*For this, select all the transactions to be canceled. Press and hold down the **Ctrl key** and **click** for selecting multiple transactions.*

*After the selection, you can click **Cancel** using right-click or via the menu bar and confirm your choice.*

*The transaction statuses will change to **Canceled**.*

14.5. Edit the order reference

This operation allows the merchant to change the order reference.

To edit the order reference of a transaction:

1. Right-click the transaction.
2. Select **Edit the order reference**.
3. Enter the new order reference.
4. Click **OK**.

14.6. Resend transaction confirmation e-mail to the buyer

To resend the transaction confirmation e-mail to the buyer in case of non-receipt or correction of the email address.

1. Look for the transaction.
2. Right-click the transaction.
3. Right-click the transaction and click **Resend transaction confirmation e-mail to the buyer**.
   The Enter e-mail address dialog box appears.
4. Enter the e-mail address.
5. Click **OK**.

14.7. Resend transaction confirmation e-mail to the merchant

To resend the transaction confirmation e-mail to the merchant:

1. Look for the transaction.
2. Right-click the transaction and click **Resend transaction confirmation e-mail to the merchant**.
   A confirmation message appears.
3. Click **OK**.

14.8. Capturing a transaction

This operation is available during test phase. It is not available in production environment.

The **Capture** option is only available for transactions that have not reached the presentation date.

To manually capture a transaction:

1. Display the tab: **Transactions in progress**
2. Select a transaction with a right-click.
3. Select **Capture manually**.
4. Confirm that you wish to definitively capture the selected transaction.

14.9. Reconciling a transaction manually

This operation allows you to manually reconcile merchant's payments from an account statement.

1. From the **Captured transactions** tab, look for the relevant transaction.
2. Right-click the transaction.
3. Select **Manual reconciliation**.
4. Click **Yes** to confirm the manual reconciliation of the selected transaction.
   The **Comment** dialog box appears.
5. Enter a comment for this reconciliation.
6. Click **OK**.

The transaction status changes to **Reconciled**.

14.10. Perform a refund

The **Make a refund** action is available when the transaction has the **presented** status.

This operation allows to re-credit the buyer's account.

The buyer's account is credited with the refunded amount, this same amount is debited from the merchant's account.

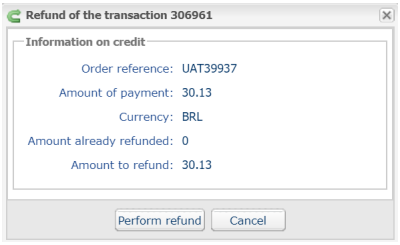
Depending on the acquirer, it is possible to refund the transaction amount partially or fully.

The refund delay available to the accounts also depend on the acquirer.

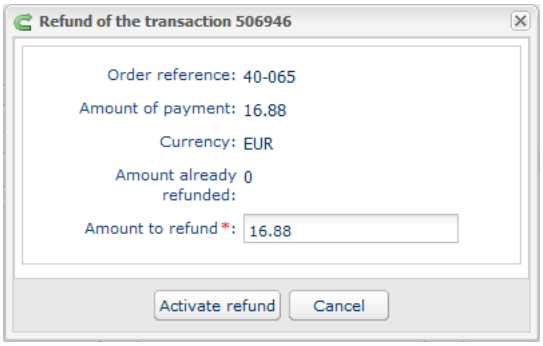
To perform a refund:

1. Display the tab **Captured transactions**
2. Select the transaction.
3. Click **Refund**.
   The dialog box **Transaction refund** appears.

Example of a full refund



Example of a partial refund



4. Enter the amount that you wish to refund.
   The entry field appears if the partial refund is possible.
5. Click **Perform refund**.

The details of the refund transaction appears.

