IMS Gateway

Integration with Google Pay

Gatewayv_2.4 - PCI v_2.6

Version Control

Version	Date	Author	Change(s)	Affected Pages
1.0	01.04.2018	IMS	Document Creation	All
1.1	08.05.2018	IMS	Reviewed with Google comments	All
1.2	10.07.2018	IMS	Reviewed with Google comments	All

Contents

1. Introduction

This document shows the option for integrating the IM PaymentSuite gateway within applications for use Google Pay.

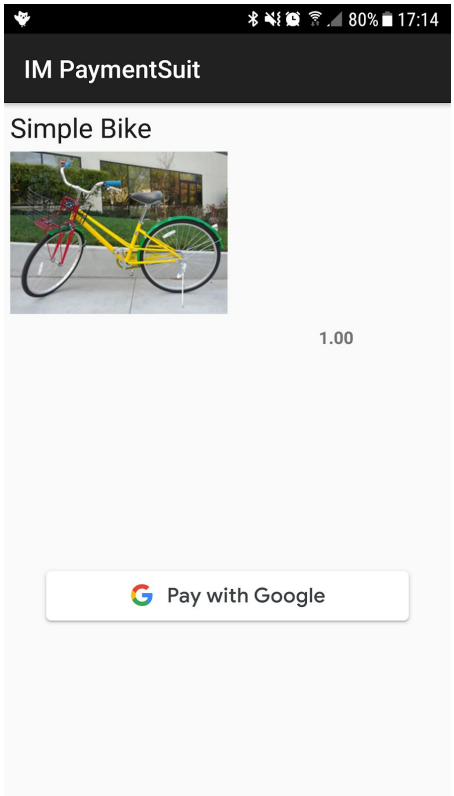
Google Pay facilitates fast and easy online purchases for your users, and eliminates manual entry of payment and shipping information. Use Google Pay to offer one-touch checkout experiences for hundreds of millions of Google users.

Google Pay can also be easily integrated for web payments, the document describes how to complete the integration through IMSolutions.

2. Basic Flow

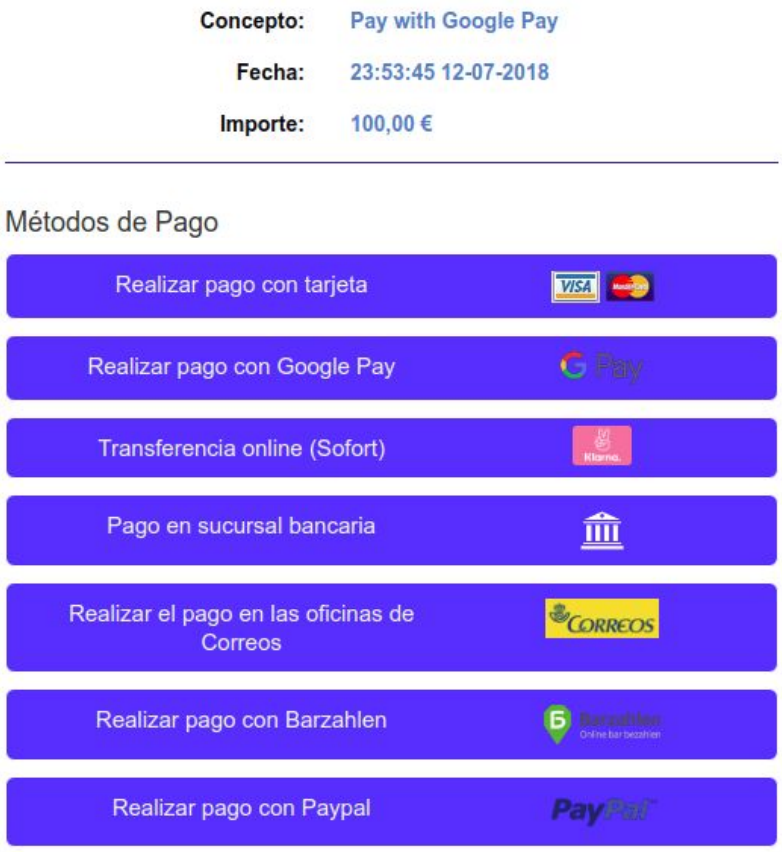
The basic flow of the sample application contains three screens that make it possible to process a payment:

1. An initial screen for the payment, where the payment is initiated, indicating the amount and payment to be processed
2. A payment screen, where the customer can select a payment card and a shipping address
3. A summary screen showing the result of the payment to the customer

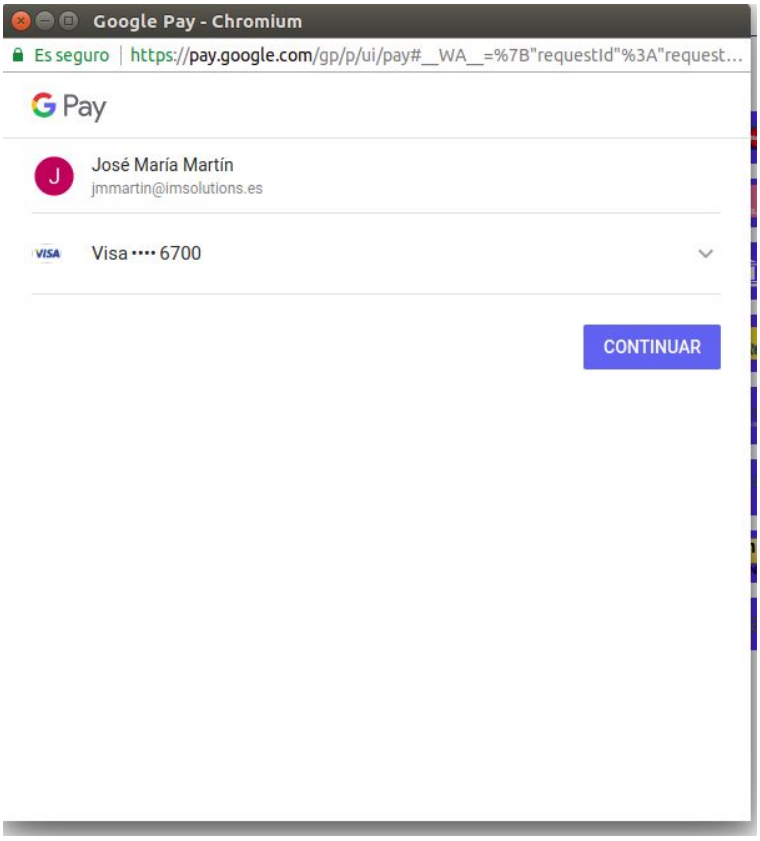


Integration through web applications

Step 1, show Google Pay button:



Step 2, select the card and pay



3. Integration with IMPaymentSuite

A simple example is given below of how to carry out the integration within a simple application.

To simplify the integration, IM has developed a library that abstracts the entire integration with the services through a series of classes which enables easy integration.

In the url (https://www.im-payment-suite.es/tienda) you can see the details of the payment gateway and request information for the integration.

3.1. Android

3.1.1. Permissions

To perform the integration, the application needs to have access permissions to Wallet Api.

To request Wallet permissions within the application, the following request must be defined in the **AndroidManifest.xml** file:

```
<meta-data
  android:name="com.google.android.gms.wallet.api.enabled"
  android:value="true" />
```

3.1.2. Configuration

It is necessary to configure all the necessary data for the environment in which the application is executed.

You must configure or create variables that are defined within the library in the Constants.java class.

3.1.2.1. Environment

Selection of the environment where the application will be executed

```
public static final int PAYMENTS_ENVIRONMENT =
WalletConstants.ENVIRONMENT_PRODUCTION;
```

For test environment you use value : *WalletConstants.ENVIRONMENT_TEST*

This value is used when the application create the PaymentClient object, this method is called when the activity is created in the start of application.

```
        public static PaymentsClient createPaymentsClient(Activity activity) {
  Wallet.WalletOptions walletOptions = new Wallet.WalletOptions.Builder()
        .setEnvironment(Constants.PAYMENTS_ENVIRONMENT)
        .build();
  return Wallet.getPaymentsClient(activity, walletOptions);
}
```

3.1.2.2. Url to IMPaymentSuite

Set the url where the application will execute the payments.

```
public static final String URL_IM = "https://imsolutionspci.es/client/rservices/gpay";
```

For test environment you use value : *"https://test.imsolutionspci.es/client/rservices/gpay"*

3.1.2.3. Merchant ID and Gateway

Set the merchant id where IMPaymentSuite will execute the payments.

You must configure the merchant id that IM will tell you for each environment

```
public static final List<Pair<String, String>> GATEWAY_TOKENIZATION_PARAMETERS
= Arrays.asList(
      Pair.create("gatewayMerchantId", "XXX")
);
```

You must replace XXX for the right value for each environment.

The name of the payment gateway is configured in constant:

```
public static final String GATEWAY_TOKENIZATION_NAME = "imsolutions";
```

These constants are used when the application calls the method to create a payment data request:

```
public static PaymentDataRequest createPaymentDataRequest(TransactionInfo
transactionInfo) {
  PaymentMethodTokenizationParameters.Builder paramsBuilder =
        PaymentMethodTokenizationParameters.newBuilder()
              .setPaymentMethodTokenizationType(

WalletConstants.PAYMENT_METHOD_TOKENIZATION_TYPE_PAYMENT_GATEWAY)
              .addParameter("gateway", Constants.GATEWAY_TOKENIZATION_NAME);
  for (Pair<String, String> param : Constants.GATEWAY_TOKENIZATION_PARAMETERS)
{
     paramsBuilder.addParameter(param.first, param.second);
  }

  return createPaymentDataRequest(transactionInfo, paramsBuilder.build());
}
```

3.1.3. Integration Google Pay with IMSolutions-Gateway

For integration, two steps are necessary:

1. Request permission and authorization the payment through Google Pay
2. Execute payment through IMPaymentSuite

The permission and authorization will be provided through the libraries of Google Pay for integration.

First, request if Google Pay is active for payment.

```
private void checkIsReadyToPay() {
  // The call to isReadyToPay is asynchronous and returns a Task. We need to provide an
  // OnCompleteListener to be triggered when the result of the call is known.
  PaymentsUtil.isReadyToPay(mPaymentsClient).addOnCompleteListener(
        new OnCompleteListener<Boolean>() {
```

```
            public void onComplete(Task<Boolean> task) {
                try {
                    boolean result = task.getResult(ApiException.class);
                    setPwgAvailable(result);
                } catch (ApiException exception) {
                    // Process error
                    Log.w("isReadyToPay failed", exception);
                }
            }
        });
}
```

If the result is ok, the application will show the payment button through Google Pay. When the user presses the Google Play button, you will receive the payment data that will be processed through the IMSolutions Gateway .

```
// This method is called when the Google Pay button is clicked.
public void requestPayment(View view) {
  // Disables the button to prevent multiple clicks.
  mPwgButton.setClickable(false);

  // The price provided to the API should include taxes and shipping.
  // This price is not displayed to the user.
  String price = PaymentsUtil.microsToString(mBikeItem.getPriceMicros() +
mShippingCost);

  TransactionInfo transaction = PaymentsUtil.createTransaction(price);
  PaymentDataRequest request = PaymentsUtil.createPaymentDataRequest(transaction);
  //PaymentDataRequest request =
PaymentsUtil.createPaymentDataRequestDirect(transaction);
  Task<PaymentData> futurePaymentData = mPaymentsClient.loadPaymentData(request);

  // Since loadPaymentData may show the UI asking the user to select a payment method, we
use
  // AutoResolveHelper to wait for the user interacting with it. Once completed,
  // onActivityResult will be called with the result.
  AutoResolveHelper.resolveTask(futurePaymentData, this,
LOAD_PAYMENT_DATA_REQUEST_CODE);
}
```

If the result is ok, payment will be processed. First, the event will be handled.

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
  switch (requestCode) {
    case LOAD_PAYMENT_DATA_REQUEST_CODE:
      switch (resultCode) {
        case Activity.RESULT_OK:
          PaymentData paymentData = PaymentData.getFromIntent(data);
          handlePaymentSuccess(paymentData);
          break;
        case Activity.RESULT_CANCELED:
          // Nothing to here normally - the user simply cancelled without selecting a
          // payment method.
          break;
        case AutoResolveHelper.RESULT_ERROR:
          Status status = AutoResolveHelper.getStatusFromIntent(data);
          handleError(status.getStatusCode());
          break;
      }

      // Re-enables the Google Pay button.
      mPwgButton.setClickable(true);
      break;
  }
}
```

When the customer clicks on Google Pay, the application returns resultCode equal to Activity.RESULT_OK, if everything is fine. The application gets the payment data in order to send them through the gateway. This data is received in the PaymentData object and handled in the handlePaymentSuccess method. In this method, we obtain all payment information and send it to IMPaymentSuit in the following method.

```
private void handlePaymentSuccess(PaymentData paymentData) {
  // PaymentMethodToken contains the payment information, as well as any additional
  // requested information, such as billing and shipping address.
  //
  // Refer to your processor's documentation on how to proceed from here.
  PaymentMethodToken token = paymentData.getPaymentMethodToken();

  // getPaymentMethodToken will only return null if PaymentMethodTokenizationParameters
was
  // not set in the PaymentRequest.
  if (token != null) {
      String billingName = paymentData.getCardInfo().getBillingAddress().getName();
     // Toast.makeText(getApplicationContext(), getString(R.string.payments_show_name,
billingName), Toast.LENGTH_LONG).show();
      Toast.makeText(this,"Processing payment. Please
wait...",Toast.LENGTH_LONG).show();

      mPwgButton.setClickable(false);
      PaymentDataDto paymentDataDto = new PaymentDataDto();
      ItemDto item = new ItemDto();
      item.setDescription(mBikeItem.getName());
      item.setUnitPrice( new
BigDecimal(PaymentsUtil.microsToString(mBikeItem.getPriceMicros())));
      item.setQuantity(1);
      paymentDataDto.addItemDto(item);


autorizationPCI(token,paymentData.getCardInfo(),paymentData.getShippingAddress(),payme
ntData.getEmail(), paymentDataDto);
      // Use token.getToken() to get the token string.
      Log.d("PaymentData", "PaymentMethodToken received");
      mPwgButton.setClickable(false);
      String msg = "Incorrect payment, there was a problem with the card";
      if(!"ERROR".equals(response)){
         msg = "Payment finished correctly with code " + response;
      }
      Toast.makeText(this,msg,Toast.LENGTH_LONG).show();
  }
}
```

The next thing will be to execute the payment against IMPaymentSuite. For the execution of the payment it is necessary to send the following information:

- PaymentMethodToken, contains the information to execute the payment
- PaymentDataDto, contains the information of the purchase data
- CardInfo, contains information of the card used
- UserAddress, contains the address information associated with the card in Google Pay
- Email, contains the email of the account used to execute the payment

All this information is provided by Google Pay through the PaymentData object that Google Pay returns when the payment is executed.

The method to invoke IMPaymenSuite must be done in background.

```
try {
   final CountDownLatch latch = new CountDownLatch(1);
   Thread uiThread = new HandlerThread("UIHandler") {
      @Override
      public void run() {
         response =
PaymentService.execute(paymentMethodToken,paymentDataDto,cardInfo,addres,email);
         latch.countDown();
      }
   };
   uiThread.start();
   latch.await();

   return response;
} catch (Exception e) {
   return false;
}
```

The call to this method will execute the payment against IMPaymentSuite, as a result it will return an object of type PaymentResponse, with the information:

- ErrorCode,error code in the invocation
- PaymentCode, code that defines the payment status
- Operation, payment identifier in IMPaymentSuite
- Amount, amount paid
- AuthorisationCode,

- CardPan,
- BankOperation, operation that identifies the payment in the bank

3.2. Web

To integrate Google Play into web payments, it is necessary to integrate the IMSolutions payment gateway (https://www.im-payment-suite.es/tienda) where the payment will be shown and managed from Google Pay

It is necessary to contact IMSolutions to request the documentation and credentials for the integration.

4. Branding guidelines

In the following links you can consult the detail to show the image of the brand.
For web integrations, management is done on the gateway

- For in-app implementations - https://developers.google.com/pay/api/android/guides/brand-guidelines
- For web implementations- https://developers.google.com/pay/api/web/guides/brand-guidelines

5. Codes

Error codes (errorCode)

Code	Description
000	The request data is correct
709	Invalid device identifier. The data received is not what was expected.
801	Incorrect encryption
901	There is no merchant id

921	Non-active merchant id
951	Protocol not supported
991,999	Errors in the application

Payment status codes (paymentCode)

Payment code	Description
000	Payment OK
100, 200	Payment KO
300	Pending payment
500	Payment KO, connection timeout
701	Payment KO for security, only admit secure payments
702	Payment KO for security, payment denied because of not secure entity
749	Payment KO for security, duplicate payment intent in same petition
789	Payment KO for security, black strategy
794	Payment KO for security, payment denied because of intents with denied card with identical or superior amount

795	Payment KO for security, payment denied for no coincide country of emission and IP
796	Payment KO for security, payment denied because country of residence does not coincide with IP
797	Payment KO for security, payment denied because of card BIN filter
798	Payment KO for security, payment denied because of card security filters
799	Payment KO for security, payment denied because of IP filters
800	Payment not finished
950	Integration error in Service invocation
900	Payment KO.
999	Payment KO.

6. Extra documentation

For more information you can visit Google's online help.

To integrate Google Pay into web applications you must follow the integration guide available in the url (https://developers.google.com/pay/api/web/guides/tutorial).

To integrate Google Pay into Android applications you must follow the integration guide available in the url (https://developers.google.com/pay/api/android/).

7. Administration

All payments are managed remotely through the IMPaymentSuite administration website, where it is possible to look up and view the details of all payments.

In addition, you can adjust the settings for the business, allowing you to:

- Security rules, max amount payment, number of payment by day….

NOTE: You will need to request the access details in order to access the website.