QPay (Buy now Pay Later)

APIs Document:

Version 1.7

March 27, 2025

ProgressSoft Corporation - Fintech solution

Document Revision History

Version	Date	Notes
1.7	March 27, 2025	Update “PS-BNPL Digital Signature workflow” section to include the merchant Id in the header request.

Table of Contents

1 Introduction

Buy Now, Pay Later (BNPL) is a type of short-term financing that allows consumers to make purchases and pay for them at a future date, interest-free. The Buy Now, Pay Later solution is a digital journey that allows customers to obtain a loan for consumer needs. The form driven flow helps the customer buy something they need/ want.

The journey is accessible through APIs listed in this document and therefore the solution integrates with any merchant for a seamless experience, for example, e-commerce websites for electronic goods or clothing websites. This is the starting point to enabling QPAY as a payment method for your customers.

To proceed with the setup, you would need to have an account on QPAY that will include the necessary details that will enable you to access the APIs.

2 QPAY Buy now pay later APIs flow:

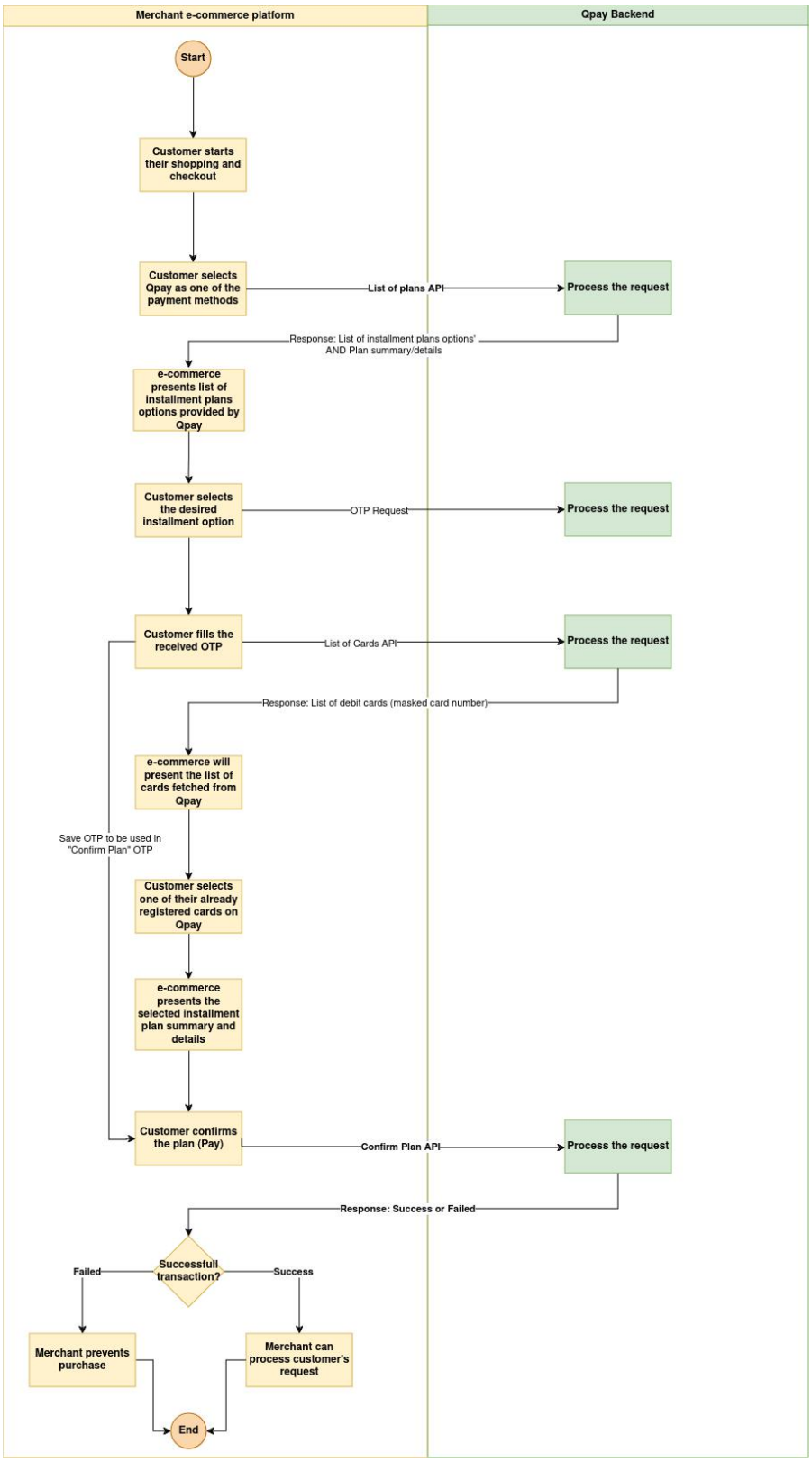

Diagram overview

The above diagram describes the APIs flow between the E-commerce and QPAY as a payment method:

- The process starts once the customer on the E-commerce chooses QPAY as a payment method on the checkout. **Note:** Customer needs to be a registered customer on Qpay application.
- The Merchant E-Commerce platform sends a request for “List of plans” API. QPAY Backend processes the request and response back with a list of the installment plans options to be presented by the E-commerce to the customer, this response will also include each installment plan details/summary. E-commerce will then present list of installment plans to the customer where the customer will select the desired installment option from the listed plans.
- The Merchant E-Commerce platform sends a request for OTP. E-commerce will display a screen to the customer where he fills the OTP he received on his mobile.
- Filling the OTP, Merchant E-Commerce platform sends a request for “List of Cards” API with the customer filled OTP is hashed and passed as input parameter to this API. QPAY Backend processes the request, verifies the OTP and response back with a list of cards and masked card numbers which are already registered. E-commerce will present to the customer a list of their defined cards for them to select the desired card to proceed with for the payment. E-Commerce will present the selected installment plan summary and details for the customer to confirm the plan and proceed with the payment.
- Once the customer confirms; e-commerce will send the request of “Confirm Plan” API with the same OTP - previously filled by the customer and saved by the E-commerce platform - as an input parameter to this API. Qpay will process the request, verifies the OTP and respond back with either a success or failed code (described in specifications below).

Depending on the response of the “Confirm Plan” API; E-commerce will either proceed with the journey of a successful transaction or prevents the customer from continuing the purchase presenting the failure reason received by Qpay.

3 QPAY BNPL APIs:

This section comprises of the specifications to be used by merchants to use QPAY exposed services to integrate with their e-commerce platform.

3.1 List of Plans APIs (list-plans)

List Of Available Plans based on Purchased Amount.

3.1.1 Request body

NO.	Field Name	Field Description	Required	Data Type
RequestHeader List of Plans				
1	lang	Customer language preference 1 for English 3 for Arabic	M	integer
2	sender	Customer mobile number used in QPAY Example: 0096812345678	M	String
3	senderType	always M	M	String
4	msgId	Unique Identifier	M	String
5	key		O	String
6	deviceId		O	String
7	otp		O	String
8	amount	Purchase amount	M	string

Example:

```
{
```

```
  "senderInfo": {
    "lang": 0,
    "sender": "string",
    "senderType": "C",
    "msgId": "string",
    "key": "string",
    "deviceId": "string",
    "otp": "string"
  },
  "amount": "string"
}
```


3.1.2 Response body

NO.	Field Name	Field Description	Data Type
1	errorCd		String
2	desc		String
3	statusCode		String
4	failureResons		String
5	principalId		integer
6	name	Name of installment option (Loan product)	String
7	description	Description of installment option (Loan product)	String
8	penalties	List of penalties associated to this installment plan (if any)	String
9	months	Number of EMI	integer
10	id	Installment plan ID	integer
11	dueDate		String
12	amount		number
13	period		integer
14	loanBalanceAfterPayment		number
15	principalId	this id will be used in the confirm plan request	integer

```
listOfPlanResponse { 
   responseInfo        common Response info { 
                          errorCd              string
                          desc                 string
                          statusCode           string
                          failureReasons        [string]

                          x-implements         ["ResponseHeader"]
                       }
   plans                [plans { 
                          name                 string
                          description          string
                          penalties             [string]
                          months               integer
                          id                   integer
                                               unique identifier for installment plan to be used in confirm plan request
                          installments          [installment { 
                                                  dueDate                string
                                                  amount                 number($double)
                                                  period                 integer
                                                  loanBalanceAfterPayment number($double)

                                                }]

                        }]
   principalId         integer
                       this id will be used in the confirm plan request

   x-implements        ["UseCaseResponse"]
}
```

Example:


```
{
  "responseInfo": {
    "errorCd": "string",
    "desc": "string",
    "statusCode": "string",
    "failureReasons": [
      "string"
    ]
  },
  "plans": [
    {
      "name": "string",
      "description": "string",
      "penalties": [
        "string"
      ],
      "months": 0,
      "id": 0,
      "installments": [
        {
          "dueDate": "string",
          "amount": 0,
          "period": 0,
          "loanBalanceAfterPayment": 0
        }
      ]
```

```
    }
  ],
  "principalId": 0
}
```

3.2 Generate OTP API (generate-otp)

This API is to generate an OTP.

3.2.1 Request body

NO.	Field Name	Field Description	Required	Data Type
RequestHeader				
1	lang	customer language preference **1** for English **3** for Arabic	M	integer
2	sender	customer mobile number used in QPAY Example: **009681234567**8	M	String
3	senderType	always M	M	String
4	msgId	Unique Identifier	M	String
storeInformation				
5	name		O	String

Example:

```
{
  "senderInfo": {
    "lang": 0,
    "sender": "string",
    "senderType": "M",
    "msgId": "string"
    },
    "storeInformation":
         {
         "name": "string"
         }
}
```

3.2.2 Response body

NO.	Field Name	Field Description	Data Type
ResponseHeader			
1	errorCd		String
2	desc		String
3	statusCode		String
4	failureReasons		String

Example:

```
{
  "responseInfo": {
```

```
    "errorCd": "string",
    "desc": "string",
    "statusCode": "string",
    "failureReasons": [
      "string"
    ]
  }
}
```

3.2.3 Hashing Algorithem

This sample code is used to hash the OTP plain text entered by the user on the mobile application, to be filled (after hashing) as input parameters to 3.2 List of Cards APIs (list-card) and 3.3 Confirm Plan APIs (confirm-plan) APIs.

Example:

```
public static String hash(String value) throws Exception {

    String encoding = "UTF-8";

    String hashingAlgorithm = "SHA-256";

    MessageDigest msgDig = MessageDigest.getInstance(hashingAlgorithm);

    msgDig.reset();

    msgDig.update(value.getBytes(Charset.forName(encoding)));

    return new String(Base64.getEncoder().encode(msgDig.digest()));

}
```

3.3 List of Cards APIs (list-card)

This API is to Get List of Cards with the filled OTP.

3.3.1 Request body

NO.	Field Name	Field Description	Required	Data Type
RequestHeader				
1	lang	customer language preference **1** for English **3** for Arabic	M	integer
2	sender	customer mobile number used in QPAY Example: **00968123456 78**	M	String
3	senderType	always M	M	String
4	msgId	Unique Identifier	M	String
5	key		O	String
6	deviceId		O	String
7	otp	Hashed OTP	M	String

Example:

```
{
  "senderInfo": {
    "lang": 0,
    "sender": "string",
    "senderType": "M",
    "msgId": "string",
    "key": "string",
    "deviceId": "string",
    "otp": "string"
  }
}
```

3.3.2 Response body

NO.	Field Name	Field Description	Data Type
ResponseHeader			
1	errorCd		String
2	desc		String
3	statusCode		String
4	failureReasons		String
cards			
5	account	to be used in confirm plan request Example:	string

		BNPL001	
6	cardNumber	Customer cards number masked except the last four digits Example: **************1234**	string
7	expiryDate		string
8	cardHolderName		string

Example:

```
{
  "responseInfo": {
    "errorCd": "string",
    "desc": "string",
    "statusCode": "string",
    "failureReasons": [
      "string"
    ]
  },
  "cards": [
    {
      "account": "string",
      "cardNumber": "string",
      "expiryDate": "string",
      "cardHolderName": "string"
    }
  ]
}
```

3.4 Confirm Plan APIs (confirm-plan)

This APIs to Confirm the plan with the selected amount and saved OTP.

3.4.1 Request body

NO.	Field Name	Field Description	Required	Data Type
RequestHeader				
senderInfo				
2	lang	customer language preference **1** for English **3** for Arabic	M	integer
3	sender	Customer mobile number used in QPAY Example: **0096812345678**	M	string
4	senderType	always **M**	M	string
5	msgId	Unique Identifier	M	String
6	key		O	String
7	deviceId		O	String
8	otp	Hashed OTP	M	String
planInfo				
9	planId	Received in ListofPlans API Response	M	integer
10	principleId	Received in ListofPlans API Response	M	integer
11	merchantId	Static ID to be provided.	M	String
12	requestId		O	String
13	account	Received in ListofCards API response	M	String
14	amount	Purchase amount	M	String

Example:

```
{
  "senderInfo": {
    "lang": 0,
    "sender": "string",
    "senderType": "M",
    "msgId": "string",
    "key": "string",
    "deviceId": "string",
    "otp": "string"
  },
  "planInfo": {
    "planId": 0,
    "principleId": 0,
    "merchantId": "string",
    "requestId": "string",
```

```
    "account": "string"
    "amount": "string"
  }
}
```

3.4.2 Response body

NO.	Field Name	Field Description	Data Type
ResponseHeader			
1	errorCd	00 (in case of success)	String
2	desc	Success (in case of success)	String
3	statusCode	1 (in case of success)	String
4	loanId	Loan Id in mPay	String
5	failureReasons	Reason in case of a failed transaction	String

In case e-commerce received any code other than “00”; this means that this transaction is rejected and the response will include the reason of failure.

Example:

```
{
  "responseInfo": {
    "errorCd": "string",
    "desc": "string",
    "statusCode": "string",
    "loanId: "string",
    "failureReasons": [
      "string"
    ]
  }
}
```

3.5 PS-BNPL Digital Signature

3.5.1 PS-BNPL Digital Signature workflow

PS-BNPL requests (Message body as is) should be signed using merchant private key.
PS-BNPL will verify the requests using the shared by merchant public key.
PS-BNPL will reject any unverified message.
PS-BNPL Will receive the token in the request header using "token" key.
PS-BNPL Will receive the merchant Id in the request header using "x-Merchant-ID" key.
PS-BNPL Will sign the response body using PS-BNPL private key - that if the merchant wants to verify the response.
merchant should verify the response body using shared by PS-BNPL public key.

The merchant shall create a key-pair and share with us the public key.

The merchant shall take our public key if they want to validate our messages.

The merchant shall send us the merchant Id that previously provided value by FintecSolutions.

3.5.2 Sample Code

This sample code is a pseudo code shared with the merchant as a sample mechanism for signing and verifying the signature in PS-BNPL

```
package com.progressoft.mpay.security;

import java.io.InputStream;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.security.Key;
import java.security.KeyStore;
import java.security.PublicKey;
import java.security.Signature;
import java.security.cert.X509Certificate;
import java.security.interfaces.RSAPrivateKey;
import java.util.Base64;
import java.util.Scanner;

import static org.apache.commons.codec.binary.Base64.decodeBase64;

public class Security {

    public boolean verifySign(String signature, String value, String algorithm, String encoding,
String keyStorePassword, String certificateAlias) {
        try {
```

```
            byte[] binaryData = value.getBytes(StandardCharsets.UTF_8);
            byte[] binarySig = decodeBase64(signature);
            X509Certificate cert = loadCertificate(keyStorePassword, certificateAlias);
            PublicKey publicKey = cert.getPublicKey();
            Signature sig = Signature.getInstance(algorithm);
            sig.initVerify(publicKey);
            sig.update(binaryData);
            return sig.verify(binarySig);
        } catch (Exception e) {
            throw new CryptographicException("Failed to verify signature", e);
        }
    }

    public String sign(String value, String algorithm, String encoding, String keyStorePassword,
String certificateAlias) {
        try {
            RSAPrivateKey privateKey = (RSAPrivateKey) loadCertificatePrivateKey(keyStorePassword,
certificateAlias);
            Signature sig = Signature.getInstance(algorithm);
            sig.initSign(privateKey);
            sig.update(value.getBytes(Charset.forName(encoding)));
            return new String(Base64.getEncoder().encode(sig.sign()));
        } catch (Exception e) {
            throw new CryptographicException("Failed to sign value", e);
        }
    }

    private X509Certificate loadCertificate(String keyStorePassword, String certificateAlias) {
        try {
            KeyStore keystore = loadKeyStore(keyStorePassword);
            return (X509Certificate) keystore.getCertificate(certificateAlias);

        } catch (Exception e) {
            throw new GenericException("Failed to load certificate, alias: " + certificateAlias, e);
        }
    }

    private KeyStore loadKeyStore(String keyStorePassword) {
        ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
        try (InputStream stream = classLoader.getResourceAsStream("keystore.jks")) {
            KeyStore keystore = KeyStore.getInstance(KeyStore.getDefaultType());
            keystore.load(stream, keyStorePassword.toCharArray());
            return keystore;
```

```
        } catch (Exception e) {
            throw new GenericException("Failed to load keystore", e);
        }
    }

    private Key loadCertificatePrivateKey(String keyStorePassword, String certificateAlias) {
        try {
            KeyStore keystore = loadKeyStore(keyStorePassword);
            return keystore.getKey(certificateAlias, keyStorePassword.toCharArray());

        } catch (Exception e) {
            throw new GenericException("Failed to load certificate private key, certificateAlias: " +
certificateAlias, e);
        }
    }

    public static void main(String[] args) {
        String certificateAlias = "mpay";
        String encoding = "UTF-8";
        String keyStorePassword = "mpayjfw";
        String hashingAlgorithm = "SHA256withRSA";
        Scanner scanner = new Scanner(System.in);
        Security security = new Security();

        System.out.println("Please enter 1 for verify and 2 for sign:");
        String code = scanner.nextLine();
        if (code.equals("1")) {
            System.out.println("Please enter the content:");
            String content = scanner.nextLine();
            System.out.println("Please enter the Token:");
            String token = scanner.nextLine();
            boolean b = security.verifySign(token, content, hashingAlgorithm, encoding,
keyStorePassword, certificateAlias);
            System.out.println("The result of verify is " + b);
        } else {
            System.out.println("Please enter the content:");
            String content = scanner.nextLine();
            String sign = security.sign(content, hashingAlgorithm, encoding, keyStorePassword,
certificateAlias);

            System.out.println("The result of sign is \n" + sign);
        }
    }
```

```
}
```