BLINKPAY

PAYIN API INTEGRATION DOCUMENT

Version 1.0

Overview

To integrate your website with a payment gateway, start by understanding your payment needs and setting up an account with the gateway provider. Ensure you implement necessary security measures and comply with data protection standards. Integrate the gateway's API into your site, test thoroughly, and then deploy your solution while monitoring its performance. Configure webhooks for real-time transaction updates and design a user-friendly interface to enhance customer experience. For access to the management portal and further assistance, contact blinkpay support.

PAYIN GATEWAY API

The Payin Gateway API allows merchants to receive payments through api.paymentsolution.in. Ensure your website integrates the payin functionalities for secure transactions.

Payin Intent Request API

Method: POST

URL: https://api.blinkpay.tech/gateway/v1/getIntent

Parameters to be posted

Parameter Name	Description	Optional/ Mandatory
apiKey	blinkpay.in assigns a unique it merchant key to each business. This key is exclusive to your account.	Mandatory
requestedOrderId	This is similar as order id it should be unique for every payment request	Mandatory
amount	Amount being requested In Rupees	Mandatory
mobileNumber	Customer mobile Number	Mandatory
username	Customer Name	Mandatory
transaction_method_id	Default is 1 for UPI transactions	Optional
signature	A cryptographic signature generated from the data provided in the request, used for ensuring data integrity and security. (refer to Appendix 1)	Mandatory
emailId	Customer's email ID	Mandatory

REQUEST:

The intent (payin) request parameter will be in JSON format as shown below:


```
{
   "apiKey": “your_api_key”,
   "amount": "10",
   "emailId": "customer@example.com",
   "mobileNumber": "1234567890",
   "username": "test",
   "transaction_method_id": "1",
   "requestedOrderId": "testorderID",
   "signature":
"d269cb254d0fb1d658c8d67e357c0c3f06a5243f92487ca64e2042142fa2792d07dd50aeac9f730
1efe184336806d9ca07cfeae836bd4c66dbead0ca0f6709ae"
}
```


ENCRYPTED REQUEST:


```
{
 "apiKey": "51dddee7-3fe4-42b1-aa3c-0000000",
"encrypted_data":
"mLKeP59KkI3NQ1n2KzIdksK5hRY5oMnZtGDLyRT9KtAkhVGWEuW+1j92NQ6wK2Ko1VW
aT2K5kWwYINv9Uw6FmjbcIPlJovOCxRUfllf7NHUKDjR9GwC9z+nZbYIJjDHRrFei8hHZgm7
TVNk1xsmwWloF7Nct2YoSn5/iq8Il7KXPhQL9vLHHMzyWRUvVgwEtKdi0zYvMgi2Sknhi3e
q0X8GZatDaDCY3L4HYuOegMdAfkhLCUGO2e5Z4XEzrKfwP8IvDoHqlfh7ax5TUMbPTrgw8t
R0KkjZJzAJuwVPDI4rx8gz4H7eUdiNGP9TPRNRMXo2nlRklTHyNjHjKPnygaEmkQ=="
 }
```


ENCRYPTED RESPONSE:


```
{ "encrypted_response":
"o78zg6AHZAnEc/0RwXfcDFpsLrOjdGJlI7Z3iNd01N3tEvcQRsyQuMOnU+vxQcIpqGr3z57P1
sdrwRxj2EnsvV+yCm7hMK2WjM35SeTqfF6vhAiFtDybiCdxjTVfGvjFYRzMn7ACjvPmtUtZnC
zQduBys93gEqlM8iC3gZi9UCcTvNhrNAWkO3ZjMGaRLN1u2ApTq6pvEm+rQFB/dF3nDFy
mmV0tx6RlL+5khOelEU/8/SP8RV1NKlHviS9BlRdBb+IgzYvihvQEfOtel3KI7dw22UnioQ+dq
EuyI1HWw7aJJtEltKoXZQ3SPYqsLFAHopdvsU+wMd+n7ronkoU8ua5Vycb+cbLv5jDViaHlq3
KAKfnNU3S+RmqOFzjp3FHQG1dpijEp0xuDE46Kly1z63hsH/4kDe4eJk16daH8asY=" }
```

RESPONSE:

On successful call to this API the response posted in JSON format will be as shown below:


```
{
   "upiIntent":
"upi://pay?pa=test@testbank&pn=test&mc=5816&tr=5525040906123008&tn=QRTxn&am=10.
00&cu=INR&mode=05&orgid=187064&catagory=01&url=https://www.testbank.com/&sign=M
EYCIQCdRCNHdNUgtP4TqvXYwfGaQTFrDjj8uWXBPpIqNpdgIhAOl1XPkfXxa/Yga9uMBEolURpPpC
W6cvoO0wGxeNUDOG",
   "orderId": "BPAYZQRU924H4PD6Z0LUGXO3F",
   "paymentRequestId": 481031,
   "requestedOrderId": "6515d169",
   "statusCode": 200
}
```


CALLBACK:

Upon a successful payment, you'll receive a callback with the following JSON payload:


```
{
   "transaction_date": "2024-07-01 13:42:14",
   "amount": "10.00",
   "transaction_status": "success",
   "response_code": "0",
   "response_message": "Transaction successful",
   "bank_ref_id": "418331180567",
   "customer_vpa": "1234567890@ybl",
   "order_id": " BPAYZQRU924H4PD6Z0LUGXO3F ",
   "requested_order_id": "0033"
}
```

After 30 minutes of initiation you will receive a failed callback with the following JSON payload


```
{
   "transaction_date": "2025-11-24T12:07:01.108",
   "amount": 2255,
   "transaction_status": "failed",
   "response_code": "400",
   "response_message": "Transaction failed",
   "orderId": " BPAYZQRU924H4PD6Z0LUGXO3F ",
   "bank_ref_id": "217055186166",
   "customer_vpa": "1234567890-3@axl",
   "requestedOrderId": "2025112411351129",
   "customer_name": " John B"
}
```

PAYIN STATUS API

Payin Payment Status API

Method: POST

URL: https://api.blinkpay.tech/status/v1/getTxnStatus

Parameters to be posted

Parameter Name	Description	Optional/ Mandatory
apiKey	blinkpay.in assigns a unique it merchant key to each business. This key is exclusive to your account.	Mandatory
orderId	OrderId Received by blinkpay	Mandatory
signature	A cryptographic signature generated from the data provided in the request, used for ensuring data integrity and security. (refer to Appendix 1)	

REQUEST:

The status request parameter will be in JSON format as shown below:


```
{
        "apiKey”: “your_api_key",
          " requestedOrderId ":" your_order_id ",
        "signature":"d7f77fdeb2662fcdbcf1a09d6a109f9f4864a494e21b7b9be5a147f495f492e2
        30194d7988cf83a690f9024ca98501c0f7ba00f87fe38ba2ab8dc2f39390e460"
}
```

TYPES OF STATUS:

• **Status**	• **Description**
• success	Status if the payment is successfully completed.
• failed	Status if the payment is not completed within 30 minutes.
• pending	Status remains in this state for up to 30 minutes while the payment is being processed.
• authorised	Status immediately after the payment is initiated.

SIGNATURE CREATION FOR STATUS API:

```
public static void main(String[] args) {
    String salt = "your_salt";
    String apiKey = "your_api_key";
    String requestedOrderId = "unique_order_id";

    // Generate signature
    String signature = getEncodedValueWithSha2(salt, apiKey,
requestedOrderId);
 System.out.println("Generated Signature: " + signature);
}
```

RESPONSE:

On successful call to this API the response posted in JSON format will be as shown below:


```
{
  "orderId": " BPAYZQRU924H4PD6Z0LUGXO3F ",
  "bank_ref_id": " 420523685154",
  "requestedOrderId": "your_unique_order_id ",
  "transaction_description": "Transaction Successful",
  "txnDate": "23-07-2024 14:43:53",
  "status": "success"
}
```


Note: If the status is authorized, it will be updated every 10 minutes. If the transaction is pending, it will fail after 30 minutes.

APPENDIX I

Signature Generation Utility

The **SignatureGenerate** class provides methods to generate HMAC-SHA512 signatures for authenticating API requests to payment gateways.

encodeWithHMACSHA2

Generates an HMAC-SHA512 hash from a given text using a specified secret key.

Code:

```
public static byte[] encodeWithHMACSHA2(String text, String keyString)
    throws java.security.NoSuchAlgorithmException,
           java.security.InvalidKeyException,
           java.io.UnsupportedEncodingException {
    java.security.Key key = new
javax.crypto.spec.SecretKeySpec(keyString.getBytes("UTF-8"),
"HMACSHA512");
    javax.crypto.Mac mac =
javax.crypto.Mac.getInstance(key.getAlgorithm());
    mac.init(key);
    return mac.doFinal(text.getBytes("UTF-8"));
}
```

ByteToHexString

Converts a byte array to a hexadecimal string, making the HMAC hash readable and usable in API requests.

Code:

```
public static String byteToHexString(byte[] bytes) {
    StringBuilder sb = new StringBuilder(bytes.length * 2);
    for (byte b : bytes) {
        int v = b & 0xff;
        if (v < 16) sb.append('0');
        sb.append(Integer.toHexString(v));
    }
    return sb.toString();
}
```

getEncodedValueWithSha2

Concatenates parameters, hashes the resulting string using HMAC-SHA512, and converts it to a hexadecimal string. This generates the final signature required for secure API interactions.

Code:

```
public static String getEncodedValueWithSha2(String hashKey, String...
params) {
    StringBuilder sb = new StringBuilder();
    for (String param : params) {
        sb.append(param);
    }
    String concatenatedParams = sb.toString();
    try {
        byte[] hmac = encodeWithHMACSHA2(concatenatedParams, hashKey);
        return byteToHexString(hmac);
    } catch (Exception e) {
        System.out.println("[Error] Unable to encode value with key: "
+ hashKey);
        e.printStackTrace();
        return null;
    }
}
```

Usage Example

Shows how to use the methods to generate a signature for API requests.

Code:

```
public static void main(String[] args) {
    // Example values
    String apiKey = "your_api_key";
    String salt = "your_salt";
    String amount = "10";
    String email = "customer@example.com";
    String mobile = "1234567890";
    String requestedOrderId = "unique_order_id";

    // Generate signature
    String signature = getEncodedValueWithSha2(salt, apiKey, amount,
email, mobile, requestedOrderId);
    System.out.println("Generated Signature: " + signature);
}
```

In the **main** method:

- **apiKey**: Unique key assigned to your account.
- **salt**: Secret key used for signature generation.
- **amount**: Payment amount.
- **email**: Customer's email address.
- **mobile**: Customer's mobile number.
- **requestedOrderId**: Unique ID for the payment request.
- **customer_vpa** : Customer Collect UPI Id. (For UPI Collect Method only.)

These methods ensure secure signature generation for API requests, maintaining data integrity and authenticity.

APPENDIX II

Encryption Decryption Utility

Code:

```
private static final String AES_SECRET_KEY = "your16ByteSecretKey";
private static final String AES_TRANSFORMATION =
"AES/ECB/PKCS5Padding";
public static String encryptAES(String data) throws Exception {
    Cipher cipher = Cipher.getInstance(AES_TRANSFORMATION);
    SecretKeySpec key = new
SecretKeySpec(AES_SECRET_KEY.getBytes("UTF-8"), "AES");
    cipher.init(Cipher.ENCRYPT_MODE, key);
    return
Base64.getEncoder().encodeToString(cipher.doFinal(data.getBytes("UTF-
8")));
}



public static String decryptAES(String encryptedData) throws Exception
{
    Cipher cipher = Cipher.getInstance(AES_TRANSFORMATION);
    SecretKeySpec key = new
SecretKeySpec(AES_SECRET_KEY.getBytes("UTF-8"), "AES");
    cipher.init(Cipher.DECRYPT_MODE, key);
    return new
String(cipher.doFinal(Base64.getDecoder().decode(encryptedData)),
"UTF8");
}
```

PHP:

```
public function encryptAES($data, $encryptkey) {
    $key = $encryptkey; $cipher = "AES-192-ECB";
    $encrypted = openssl_encrypt($data, $cipher, $key,
OPENSSL_RAW_DATA);
    return base64_encode($encrypted);
}

public function decryptAES($encryptedData, $encryptkey) {
    $key = $encryptkey; $cipher = "AES-192-ECB";
    $decodedData = base64_decode($encryptedData);
    $decrypted = openssl_decrypt($decodedData, $cipher, $key,
OPENSSL_RAW_DATA);
    return $decrypted;
}
```

Usage Example:

```
 public static void main(String[] args) {
    try {
       String apiKey = "your_api_key";
       String salt = "your_salt";
       String amount = "10";
       String email = "customer@example.com";
       String mobile = "1234567890";
       String requestedOrderId = "unique_order_id";
       String signature = getEncodedValueWithSha2(salt, apiKey,
amount, email, mobile, requestedOrderId);
       JSONObject payloadJson = new JSONObject();
       payloadJson.put("amount", amount);
       payloadJson.put("apiKey", apiKey);
       payloadJson.put("emailId", email);
       payloadJson.put("mobileNumber", mobile);
       payloadJson.put("requestedOrderId", requestedOrderId);
       payloadJson.put("signature", signature);
       payloadJson.put("username", "test");
       String payload = payloadJson.toString();
       String encrypted = encryptAES(payload);
       String decrypted = decryptAES(encrypted);
       System.out.println("Signature: " + signature);
       System.out.println("Encrypted: " + encrypted);
       System.out.println("Decrypted: " + decrypted);
    } catch (Exception e) {
       e.printStackTrace();
    }
}
```

HTTP Status Codes

HTTP Status Code	Message	Description
200	OK / Success	The request was successful, and the server returned the expected response.
400	Bad Request / Invalid Input	The request was malformed or contained invalid data.
403	Forbidden / IP Not Whitelisted	The request is not authorized because the IP address is not whitelisted.
404	Not Found / Request URL Not Found	The requested resource or URL does not exist on the server.
500	Internal Server Error / Server Issue	The server encountered an error while processing the request.
503	Service Unavailable / Server Downtime	The server is currently unable to handle the request due to downtime or maintenance.