

SKIPCASH

SKIPCASH INTEGRATION MANUAL v1

About this guide	3
Audience and purpose	3
Web Site Requirements	3
Customer support	3
1. Process overview.....	4
2. Card Checkout API.....	5
2.1 Create an online payment.....	5
2.1.1 The Authorization header	7
2.1.2 Examples of request and responses	10
2.2 Load the detail of the Online payment	12
2.2.1 The Authorization header	12
2.2.2 Examples of request and responses	12
3. Card Checkout Configuration	14
3.1 Online Payments page	14
3.2 Sandbox -> Credentials page.....	15
3.3 Sandbox -> API Calls page	15
3.4 Sandbox -> Webhook Events page.....	16
3.5 Sandbox -> Webhooks Simulator page	16
3.6 Production -> Credentials page.....	17
4. Webhooks	18
4.1 WebHooks URL configuration	18
4.2 Implementing WebHooks	19
4.2.1 The Authorization header	20
4.2.2 Examples of request and responses	22
4.3 Debugging WebHooks.....	22
4.4 Testing WebHooks	23
5. The recommended integration process.....	24
5.1 Backend.....	24
5.2 Frontend.....	25
6. Going to Production	26

ABOUT THIS GUIDE

Audience and purpose

This guide is written for Merchants who want to customize and control their own customer checkout experience. Using the Card Checkout API requires moderate programming skills.

Web Site Requirements

Your website must meet the following **requirements**:

- It must have a shopping cart or customer order creation software.
- The IT infrastructure must be Public Key Infrastructure (PKI) enabled to use SSL-based form POST submissions. (secure transfer over https)
- The IT infrastructure must be capable of digitally signing data prior to submission to SkipCash API. (as described in section 2.1.1 and 2.2.1)

Customer support

For support information about any SkipCash service, kindly contact the Support Center:
support@skipcash.com

1. PROCESS OVERVIEW

If the Merchants want to implement the Online payments, they supposed to have:

- An active Merchant account in PROD environment
- An access to the SkipCash Merchant portal in PROD environment
- Activated account for the Online payments
- Generated private key (more details in configuration section)

SkipCash supports **two** primary use-cases:

Process overview for shoppers buying on the website:

1. Shopper is on the Merchant website and wants to buy the product.
2. Shopper clicks on the „Pay with SkipCash“ button on the Merchant checkout page.
3. The Merchant sends a new request to theirs BE server to create the online payment.
4. The Merchant receives the payment URL. The Merchant can either open a new browser window or redirect to this URL for payment processing. SkipCash collects all necessary information and processes the payment.
5. The browser window is closed or the current window is redirected back to configured return URL by the Merchant in the Merchant Portal. The payment result can be loaded again to get the final status and the Merchant can proceed accordingly.

Process overview for shoppers buying on 3rd party mobile application:

This is similar to the process described above:

1. The Merchant must create the Online payment on theirs BE server.
2. The Merchant will receive the payment URL.
3. They open a web view with the received URL.
4. After the payment is finished, they detect the browser closing and load the result of the payment.

2. CARD CHECKOUT API

SkipCash provides **2 REST endpoints** for facilitating the Online payments:

- Create a payment (POST */api/v1/payments*)
- Get a detail of a payment (GET */api/v1/payments/{id}*)

To call these endpoints, the following conditions must be met:

- Enabled Online payments
- Generated one private key

The private key is used to authenticate both of these endpoints. It consists of:

- key ID
- key secret

The key ID is a public identifier of the private key. SkipCash uses it for the identification of the key and the Merchant. Every Merchant has an additional public identifier called Client ID.

The key secret is used to calculate the signature from the request body in a create a payment endpoint, and Client ID is validated in getting the detail of a payment request.

Please note that the key secret must be stored securely on the Merchant server and never send to the frontend (either browser or mobile app). This means, that when creating a new online payment, it has to be done from the backend of the Merchant server.

If the private key is compromised, the 3rd party can create online payments on behalf of the Merchant.

2.1 Create an Online payment

REST endpoint for the Online payment creation:

POST */api/v1/payments*

```
{  
  "uid": "string",  
  "keyId": "uuid",  
  "amount": "string",  
  "firstName": "string",  
  "lastName": "string",  
  "phone": "string",
```

```

"email": "string",
"street": "string",
"city": "string",
"state": "string",
"country": "string",
"postalCode": "string",
"transactionId": "string",
"custom1": "string"
}

```

The description of the fields in the request body:

- **uid** – A random string, usually generating a new UUID is fine. Helps to randomize calculated signature.
- **keyId** – the ID of the generated private key
- **amount** – the transaction amount. It is a string. The number must be separated by a dot and has a max of 2 decimal places. Examples of valid values: „10“, „10.2“, „10.24“. Example of invalid values: „.10“, „10.“, „10,1“, „10.123“.
- **firstName** – the shopper first name
- **lastName** – the shopper last name
- **phone** – the shopper phone number (with or without country code)
- **email** – the shopper email
- **street, city, state, country, postalCode** – the shopper address information
- **transactionId** – the merchant internal order ID. Useful when processing webhooks from SkipCash.
- **custom1** – the Merchant custom value, can be used in any way by the merchant

Fields validation in the request:

Property Name	Mandatory / Optional	Restriction
uid	M	uuid (random)
keyId	M	uuid
amount	M	string (with dot as separator) max 2 decimal numbers
firstName	M	Max length 60
lastName	M	Max length 60
phone	M	Max length 15
email	M	Max length 255


```
"uid": "01916FC2-411B-4465-A3A8-F7699AD2F757",  
"firstName": "Peter",  
"lastName": "Green",  
"email": "green@test.sk"  
}
```

The merchant has the following configuration:

Client ID: [REDACTED]
Key ID: [REDACTED]
Key Secret:

[REDACTED]
[REDACTED]
[REDACTED]
[REDACTED]
[REDACTED]
[REDACTED]
[REDACTED]

Combining all the request fields into key=value line separated by comma:

Uid=01916FC2-411B-4465-A3A8-F7699AD2F757,KeyId=[REDACTED]
[REDACTED],Amount=11,FirstName=Peter,LastName=Green,Email=green@test.sk

Encrypt using HMACSHA256 and convert to base 64:

KEORv9P7MacIbhbV4uCEsgS4TcKIWOI0sXYPZCReiGg=

Example code in c#:

```
public static string CalculateSignature(request, string secretKey)  
{  
    var data = BuildData(request);  
    UTF8Encoding encoding = new UTF8Encoding();  
    byte[] keyByte = encoding.GetBytes(secretKey);  
  
    var hmacsha256 = new HMACSHA256(keyByte);  
    byte[] messageBytes = encoding.GetBytes(data);  
    return Convert.ToBase64String(hmacsha256.ComputeHash(messageBytes));  
}
```



```
}
```

```
private string BuildData(request)
```

```
{
```

```
    var list = new List<string>();
```

```
    AppendData(list, "Uid", request.Uid);
```

```
    AppendData(list, "KeyId", request.KeyId.ToString());
```

```
    AppendData(list, "Amount", request.Amount);
```

```
    AppendData(list, "FirstName", request.FirstName);
```

```
    AppendData(list, "LastName", request.LastName);
```

```
    AppendData(list, "Phone", request.Phone);
```

```
    AppendData(list, "Email", request.Email);
```

```
    AppendData(list, "Street", request.Street);
```

```
    AppendData(list, "City", request.City);
```

```
    AppendData(list, "State", request.State);
```

```
    AppendData(list, "Country", request.Country);
```

```
    AppendData(list, "PostalCode", request.PostalCode);
```

```
    AppendData(list, "TransactionId", request.TransactionId);
```

```
    AppendData(list, "Custom1", request.Custom1);
```

```
    return string.Join(',', list);
```

```
}
```

```
private void AppendData(List<string> list, string name, string value)
```

```
{
```

```
    if (!string.IsNullOrEmpty(value))
```

```
    {
```

```
        list.Add($"{name}={value}");
```

```
    }
```

```
}
```

2.1.2 Examples of request and responses

The Merchant configuration used in these example requests is the same as in the previous section.

Example 1 – Request with the mandatory fields:

POST /api/v1/payments

Authorization: rKE6vUxneWOflG/Tx2FP9dSTm+QVnn0kCWu7njOZidk=

```
{
  "amount": "15.25",
  "keyId": "[REDACTED]",
  "uid": "1A7447ED-BE99-4385-A814-91292CFB8003",
  "firstName": "John",
  "lastName": "Doe",
  "email": "john-doe@test.sk"
}
```

Example response:

```
{
  "resultObj": {
    "id": "f0f18db8-ae45-4be2-a155-9f7f0714874b",
    "statusId": 0,
    "created": "2021-06-30T13:11:01Z",
    "payUrl": "https://skipcashtest.azurewebsites.net/pay/f0f18db8-ae45-4be2-a155-9f7f0714874b",
    "amount": 15.25,
    "currency": "QAR",
    "transactionId": null,
    "custom1": null,
    "visaId": null,
    "status": "new"
  },
  "returnCode": 200,
  "errorCode": 0,
  "errorMessage": null,
  "error": null,
  "validationErrors": null,
  "hasError": false,
  "hasValidationError": false
}
```

Example 2 – Request with the optional fields:

POST /api/v1/payments

Authorization: yRXjFyoyh8kLSqdgb/UerWFCBp5cSBTGmbdxdpParEzo=

```
{
  "amount": "19",
  "keyId": " ",
  "uid": "9FE2928B-0A1C-47CC-AF60-A9B14134B510",
  "firstName": "John",
  "lastName": "Doe",
  "email": "john-doe@test.sk",
  "phone": "+974000000001",
  "street": "Al Samriya St 10",
  "country": "QA",
  "city": "Doha",
  "transactionId": "custom-internal-id",
  "custom1": "test"
}
```

Example response:

```
{
  "resultObj": {
    "id": "770bd18b-a505-468b-8de1-ac7f9555dc79",
    "statusId": 0,
    "created": "2021-06-30T13:33:52Z",
    "payUrl": "https://skipcashtest.azurewebsites.net/pay/770bd18b-a505-468b-8de1-ac7f9555dc79",
    "amount": 19,
    "currency": "QAR",
    "transactionId": "custom-internal-id",
    "custom1": "test",
    "visalId": null,
    "status": "new"
  },
  "returnCode": 200,
  "errorCode": 0,
  "errorMessage": null,
  "error": null,
  "validationErrors": null,
  "hasError": false,
  "hasValidationError": false
}
```

2.2 Load the detail of the Online payment

REST endpoint for loading the detail of the Online payment:

GET /api/v1/payments/{id}

- **id** – the internal SkipCash ID of the payment

Response fields:

- **id** – the internal SkipCash ID of the payment
- **statusId** – the payment status, 0 – new, 1 – pending, 2 – paid, 3 – canceled, 4 – failed, 5 – rejected, 6 – refunded, 7 – pending refund, 8 – refund failed
- **created** – the created date in UTC timezone
- **payUrl** – the full URL for payments
- **amount** – the payment amount
- **currency** – the payment currency, always QAR
- **transactionId** – the merchant order ID
- **custom1** – the merchant custom identifier
- **visalId** – the internal CyberSource payment ID. Record this ID into your database for reconciliation purposes.
- **status** – the text representation of statusId

2.2.1 The Authorization header

The header is simple in this case. Sending the Client ID of the Merchant is sufficient.

2.2.2 Examples of request and responses

Example 1 – detail of transaction with id 770bd18b-a505-468b-8de1-ac7f9555dc79

GET /api/v1/payments/770bd18b-a505-468b-8de1-ac7f9555dc79

Example response:

```
{
  "resultObj": {
    "id": "770bd18b-a505-468b-8de1-ac7f9555dc79",
    "statusId": 0,
    "created": "2021-06-30T13:33:52Z",
    "payUrl": "https://skipcashtest.azurewebsites.net/pay/770bd18b-a505-468b-8de1-ac7f9555dc79",
    "amount": 19.000,
  }
}
```

```
"currency": "QAR",
"transactionId": "custom-internal-id",
"custom1": "test",
"visaId": null,
"status": "new"
},
"returnCode": 200,
"errorCode": 0,
"errorMessage": null,
"error": null,
"validationErrors": null,
"hasError": false,
"hasValidationError": false
}
```

3. CARD CHECKOUT CONFIGURATION

3.1 Online Payments page

The Merchant can enable or disable the Online payments separately for the TEST (Sandbox) and PRODUCTION environment.

When the Online payments are disabled, all Merchant keys are disabled automatically as well.

Enable online payments for the Sandbox (TEST) environment

- Click on the button „Enable Sandbox“

Disable online payments for the Sandbox (TEST) environment

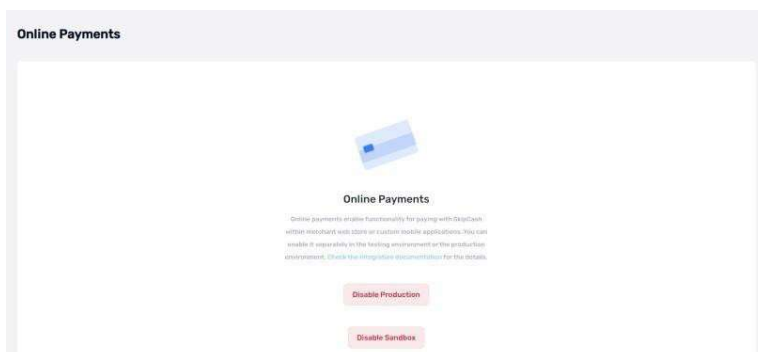
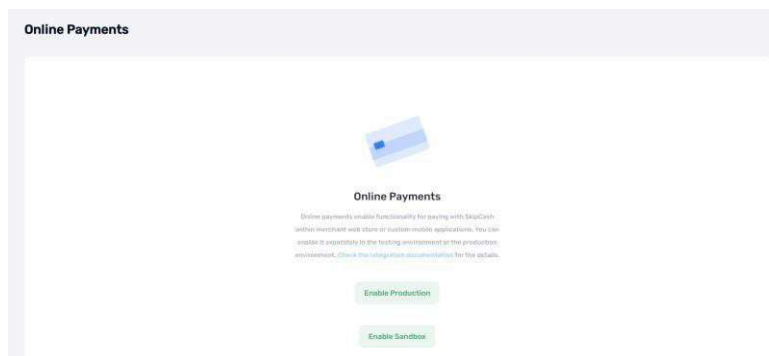
- Click on the button „Disable Sandbox“

Enable online payments for the Production environment

- Click on the button „Enable Production“

Disable online payments for the Production environment

- Click on the button „Disable Production“



3.2 Sandbox -> Credentials page

Sandbox Credentials API Calls Webhook Events Webhooks Simulator

Generate a New Secret Key

Client ID: 3121b794-a463-4440-a8f9-235cbefc19d0

Webhook Key: 54d236b5-bfba-44e5-a978-6a0e4482006f

Webhook URL

Return URL

Save

Key Id	Key Secret	Status	Type
--------	------------	--------	------

Copy basic configuration:

You can copy the Client ID and Webhook Key values to your Merchant configuration. The respective values will not change.

Configure the webhook URL (optional):

1. Type your Merchant URL of the endpoint for processing webhooks from SkipCash
2. Click on the button „Save“

Configure the return URL (optional):

1. Type your return URL of the endpoint for returning from SkipCash back to Merchant.
2. Click on the button „Save“

When the return URL is configured, the payment flow changes accordingly:

Once the payment is finished, instead of automatically closing the browser window, the SkipCash is redirecting back to the respective configured return URL.

Additional parameters like id, statusId, and status of the processed transaction are added. It is highly important to call the SkipCash REST endpoint for the transaction detail and verify the status result of the processed transaction. This is for security reasons, since URL parameters can be altered. can be altered by the shopper.

It is highly important to call the SkipCash REST endpoint for the transaction detail after and verify the status result of the processed transaction

Create a private key:

1. Click on the button „Generate a New Secret Key“
2. The new record is created and shown in the table
3. Store the key ID and key secret in your Merchant configuration

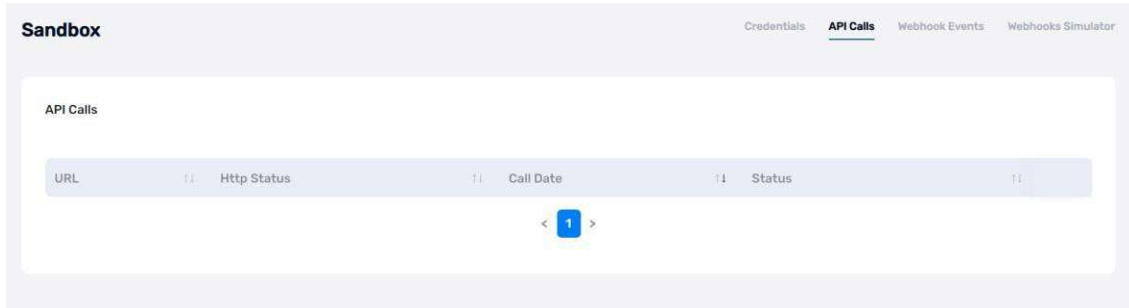
Usually, one private key per Merchant is sufficient. You can periodically replace your old keys to increase security:

Replace a private key:

1. Click on the button „Generate a New Secret Key“
2. Replace the key ID and key secret in your Merchant configuration
3. When the new key is used in production, you can disable your old key

3.2 Sandbox -> API Calls page

The table contains the list of all calls from the Merchant server to the SkipCash server. Useful for debugging and testing the public endpoints. The details of each request are in the separate modal window after clicking on the link „Detail“.

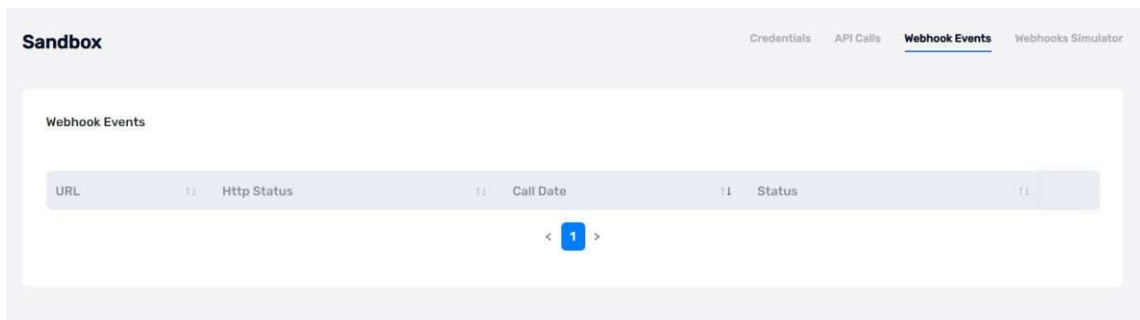


The screenshot shows the 'Sandbox' interface with the 'API Calls' tab selected. It features a table with columns: URL, Http Status, Call Date, and Status. A pagination control at the bottom indicates 1 item is displayed.

URL	Http Status	Call Date	Status
-----	-------------	-----------	--------

3.3 Sandbox -> Webhook Events page

The table contains the list of all webhook calls from the SkipCash server to the Merchant server. Useful for debugging and testing the webhook processing.



The screenshot shows the 'Sandbox' interface with the 'Webhook Events' tab selected. It features a table with columns: URL, Http Status, Call Date, and Status. A pagination control at the bottom indicates 1 item is displayed.

URL	Http Status	Call Date	Status
-----	-------------	-----------	--------

3.4 Sandbox -> Webhooks Simulator page

Webhook simulator page supports sending the specified webhook request from SkipCash to the Merchant server. Useful to test the end processing with webhooks.

- **paymentId** – internal SkipCash transaction ID
- **event type** – specify the event you want to test (Success, Failure, Cancel)
- **amount** – the amount on the transactions
- **transactionId** – internal merchant order ID

Sandbox

CredentialsAPI CallsWebhook EventsWebhooks Simulator

Webhooks Simulator

Payment ID

Event Type

Amount

Transaction ID

Success payment

Send

How to set up webhooks is described in detail in section 4.

3.5 Production -> Credentials page

The equal page as in Sanbox page (3.2 Sandbox -> Credentials page), just for the Production environment.

Credentials

Generate a New Secret Key

Client ID

Webhook Key

af031497-2e65-4e9e-9cbd-0b894fec2d06

b1aa9add-84e3-40eb-acd5-f0463e7382a5

Webhook URL

Save

Id	Secret Key	Status	Type
< 1 >			

4. WEBHOOKS

The SkipCash server will notify the Merchant server about the payment status when the **Webhook** is properly configured within the Merchant Portal.

By using Webhooks it is possible to notify the Merchant page when the shopper closes the browser window right after the payment is completed, hence the Merchant is notified and able to finish the respective payment from the background.

Kindly note, that for the one single Merchant order, multiple webhook calls can be triggered from the SkipCash.

For example when:

- The shopper clicks on the Pay button and immediately closes the browser window. This action may be repeated several times. Every time a new transaction will be created for the same merchant order. All created unprocessed transactions will be automatically canceled within 1 hour. The merchant will be notified of the cancelation event multiple times.
- The Shopper clicks on the Pay button, confirms the payment, but the payment fails. The Shopper will close the browser window and try the same action again. For each failure, the Merchant will be called with a Failure event. When failure happens, SkipCash will not finish the existing original payment to preserve the payment link. Instead, an exact transaction copy with a new generated ID will be created and stored with status Fail. This means, that the payment ID in the webhook call will be different and unknown to the Merchant (if the Merchant stores our original internal ID). In this case, the best is to ignore this webhook call.

4.1 WebHooks URL configuration

To use the webhooks system, the **Webhooks URL** must be configured in Sandbox - Credentials first:

1. Go to Sandbox Page
2. Type URL for webhook processing. The SkipCash will call this URL for every transaction status change.
3. Click on „Save“
4. SkipCash will send webhooks to the Merchant server.

4.2 Implementing WebHooks

When the transaction finishes, SkipCash will call the Merchant server:

POST {configured WebHooks Url}

Authorization: {calculated hash}

```
{  
  "paymentId": "uuid",  
  "amount": "string",  
  "statusId": "number",  
  "transactionId": "string",  
  "custom1": "string",  
  "visalId": "string",  
}
```

The request fields:

- **paymentId** – the internal SkipCash payment ID
- **amount** – the transaction amount
- **statusId** – the transaction status
- **transactionId** – the merchant order ID
- **custom1** – the merchant custom text
- **visalId** – the internal Cybersource transaction ID

Example code in c#:

```
public static string CalculateWebHookSignature(WebHookRequest request, string secretKey)
{
    var data = BuildWebHookData(request);
    UTF8Encoding encoding = new UTF8Encoding();
    byte[] keyByte = encoding.GetBytes(secretKey);

    var hmacsha256 = new HMACSHA256(keyByte);
    byte[] messageBytes = encoding.GetBytes(data);
    return Convert.ToBase64String(hmacsha256.ComputeHash(messageBytes));
}

private static string BuildWebHookData(WebHookRequest request)
{
    var list = new List<string>();
    AppendData(list, "PaymentId", request.PaymentId.ToString());
    AppendData(list, "Amount", request.Amount);
    AppendData(list, "StatusId", request.StatusId.ToString());
    AppendData(list, "TransactionId", request.TransactionId);
    AppendData(list, "Custom1", request.Custom1);
    AppendData(list, "VisId", request.VisId);
    return string.Join(',', list);
}

private void AppendData(List<string> list, string name, string value)
{
    if (!string.IsNullOrEmpty(value))
    {
        list.Add($"{name}={value}");
    }
}
```

4.2.2 Examples of request and responses

The Merchant configuration (WebHook Key) used in these example requests is the same as in the previous section 4.2.1.

Example – Sample request:

```
POST {configured WebhookUrl}
Authorization: rRdV4MzLuEvtNUX1OwvqLCg6AqbB/jOCZhCiMoKYJI8=
{
  "paymentId": "cee9db13-dc01-4bc6-a216-684f3ee05d95",
  "amount": "50.00",
  "statusId": 2,
  "transactionId": null,
  "custom1": null,
  "visaId": "6251291659776351404004",
}
```

Example response:

It will depend on the Merchant's response. SkipCash will look only for HTTP status. If the status is 200, the webhook call is counted as a success. Otherwise as a failure, and SkipCash will call the Merchant with the same request later.

4.3 Debugging WebHooks

Every call from the SkipCash server to the Merchant server is logged. You can review the full request and response logged in Webhook Events on Sandbox Page.

SkipCash will try to call the Merchant server 3 times in total. The Merchant must return HTTP status 200. Otherwise, it is counted as a failure, and SkipCash will try to call the endpoint later (for a total of 3 times).

The timeout for the request is 10 seconds, after this time the request is also counted as a failure. The retry times are as following:

- Immediately after finishing the transaction
- 1 hour later
- 1 day later

4.4 Testing WebHooks

The Webhooks can be tested on the Webhooks Simulator page in Sandbox. This is possible only when the WebHooks URL is configured.

The 4 parameters should be configured:

- **Payment ID** – this is our internal ID. Useful when you have already created some unfinished transactions and want to test the end processing.
- **Event Type** – the test scenario, successful, failure or cancel. For each, there is an alternative with the wrong header attribute in the request, which should be always rejected, as it means that it is not from SkipCash. The SkipCash will calculate the signature with Webhook Key for the Merchant. The calculation is the same as when the Merchant creates the transaction on theSkipCash server.
- **Amount** – the amount on the transaction
- **Transaction ID** – Merchant internal transaction ID

After clicking on the Send button, the WebHooks URL endpoint is called with specified data. The result is logged in the Webhook Events tab.

We recommend the following process:

1. Create an order within the merchant TEST environment
2. Create an Online payment with transactionId set to your order id
3. In Webhook Simulator fill paymentId and transactionId and click on Send button
4. Do this for the following event types: Success payment, Failure payment, and Cancel payment. Perhaps only for Success payment action is required, the rest can be ignored.
5. Once everything is working as expected, you can implement events with the wrong keys. You need to validate the authorization header, otherwise, you can't be sure if SkipCash sends the webhook.
6. Test the rest of event types: Success payment signed with the wrong key, Failure payment signed with the wrong key, Cancel event signed with the wrong key.

5. THE RECOMMENDED INTEGRATION PROCESS

5.1 Backend

1. Setup online payments in the Merchant portal as described in 3.1 and 3.2.

- a. Enable Online payments for the TEST environment by clicking on 'Enable Sandbox'
- b. Create one private key in Sandbox

2. Create basic configuration in the Merchant application:

- **SkipCashUrl** – the URL to SkipCash TEST environment is as following:
<https://skipcashtest.azurewebsites.net>
- **SkipCashClientId** – the Client Id, from Sandbox -> Credentials page
- **SkipCashKeyId** – the private key ID, from Sandbox -> Credentials page, column Id
- **SkipCashKeySecret** – the private key Secret, from Sandbox -> Credentials page, column Secret Key
- **SkipCashWebhookKey** – the private webhook key, from Sandbox -> Credentials page

3. Implement create Online payment functionality:

Described process in section 2.1. The hardest part is calculating the correct authorization header from the request body.

4. Implement get online payment functionality:

Described process in section 2.2.

5. Create and test the endpoint for processing webhooks from SkipCash:

Described process in section 4.

5.2 Frontend

1. **Show the button „Pay with SkipCash“ on the checkout page.**
2. **After the Shopper clicks the „Pay with SkipCash“ button the following process is required:**
 - a. Open a web view/browser window with the URL „about:blank“. It must be an immediate action after clicking on this button.
 - b. Create a backend request to your server, where a new online payment is created. Return the payUrl to the FE.
 - c. Change the URL of opened webview/browser window to the URL returned from SkipCash.
 - d. For Apple Pay, if a WebView is used, some additional configuration maybe required by the WebView library/package. This is dependent on the WebView library.
 - e. In the meantime periodically check if the browser window is still opened
 - f. When the Shopper pays, the browser window is closed by the script automatically.
 - g. When the browser window is closed, the transaction finished, either successfully or not. Or was manually closed by the Shopper.
 - h. Create a call to your backend server to check the final status of the transaction.
 - i. If the transaction was not paid, there is nothing to do. The shopper can pick another form of payment or try again with SkipCash.

6. GOING TO PRODUCTION

When everything is working as expected in the TEST environment, you can proceed to go live into Production. To do so, you should change your backend configuration:

1. Setup online payments in the Merchant portal as described in 3.1 and 3.6.

- a. Enable online payments for the PRODUCTION environment by clicking on 'Enable Production'
- b. Create one private key in Production

2. Change basic configuration in the Merchant application:

- **SkipCashUrl** – the URL for Production environment to SkipCash is as following:
<https://api.skipcash.app>
- **SkipCashClientId** – the Client Id, from Production -> Credentials page
- **SkipCashKeyId** – the private key ID, from Production -> Credentials page, column Id
- **SkipCashKeySecret** – the private key Secret, from Production -> Credentials page, column Secret Key
- **SkipCashWebhookKey** – the private webhook key, from Production -> Credentials page

3. Set a new production webhook URL in Production – Webhook URL (optional)

This ensures correct backend processing. After the transaction finishes, SkipCash will call the Merchant server with the result operation. Helps with a corner case such as internet loss connection or shopper closes the browser window prematurely.

4. Set a new production return URL (optional)

As mentioned earlier, the payment flow will change, instead of closing the window, SkipCash will redirect back to the Merchant page. Only set this value if you want to change the payment flow.

5. Frontend changes of the Merchant application:

Depends on the Merchant implementation. There is probably nothing needed to be changed in Frontend of the Merchant application.